Intersecção de Caminhos mais Longos em Produtos de Grafos

Gabriel Matheus Faria de Almeida[1], **Elisângela Silva Dias**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Câmpus Samambaia – 74690-900 – Goiânia – GO – Brasil

{gabrielmatheus,elisangela}@inf.ufg.br

Abstract. *The problem of longest paths in graphs, began to be discussed in 1966 from a question raised by Gallai, which questions whether any connected graph has at least one vertex that appears in all the longest paths. In this work, we studied the longest paths in chordal graphs, weighted graphs, trees and complementary prisms graphs and we obtained theoretical results and some factorial time algorithms, trying to understand the problem and the behavior of the longest paths in some types of graphs.*

Resumo. *O problema de caminhos mais longos começou a ser discutido por volta de 1966 a partir de uma questão levantada por Gallai, que questiona se todo grafo conexo tem pelo menos um vértice que aparece em todos os caminhos mais longos. Neste trabalho, foram estudados os caminhos mais longos em produtos de grafos, grafos cordais, grafos ponderados, árvores e grafos prismas complementares e foram obtidos resultados teóricos e alguns algoritmos de tempo fatorial buscando entender o problema e o comportamento dos caminhos mais longos nos diferentes tipos de grafos.*

1. Introdução

Um grafo $G = (V(G), E(G))$ consiste em $V(G)$, um conjunto não vazio de *vértices*, e $E(G)$, um conjunto de *arestas*, que são pares não ordenados de vértices. Cada aresta contém um ou mais vértices associados a ela, chamados de extremidades. Sejam v_1, v_2 vértices de G e e uma aresta que os associa; diz-se que e liga ou conecta v_1 e v_2. Os vértices v_1, v_2 são ditos *adjacentes* em G, pois existe $e = v_1v_2 \in E(G)$. Diz-se também que e é incidente a v_1 e v_2.

Um grafo não orientado é dito *conexo* se existir um caminho entre qualquer par de vértices distintos no grafo, caso contrário diz-se que o grafo é *desconexo*. Um vértice é dito *isolado* quando não tem outros vértices adjacentes a ele. Uma aresta é chamada de *laço* quando tem o vértice de saída igual ao vértice de chegada, isto é, uma aresta que liga o vértice a ele mesmo. *Arestas múltiplas* são arestas distintas incidentes ao mesmo par de vértices e um grafo é dito *simples* quando não contém laços em seus vértices nem arestas múltiplas.

Um grafo orientado é o grafo em que as arestas possuem um vértice de partida e um vértice de chegada, denotadas por uma seta e a passagem pela aresta deve obedecer a direção e o sentido que a aresta representa. No grafo não orientado as arestas não possuem orientação e são denotadas por segmentos de retas, ou seja, pode-se ir e voltar na aresta sem a necessidade de obedecer uma orientação.

O *grau* de um vértice $v \in V(G)$, denotado por $g(v)$, é o número de arestas incidentes a ele. Observe que um laço em um vértice contribui duas vezes para o seu grau. Por exemplo, na Figura 1, g(a) = 2, g(d) = 4 e g(e) = 3.

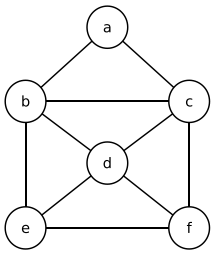


Figura 1. Exemplo de grafo simples não orientado.

Um *caminho simples* é uma sequência finita de vértices $\{v_1, v_2, ..., v_n\}$ denotado por P_n, tal que $v_i v_{(i+1)} \in E(G)$ para todo $i \in \{1, ..., n-1\}$ e sem vértices repetidos na sequência, isto é, $v_i \neq v_j$ para todo $j \in \{1, ..., n\}$ com $j \neq i$. Um ciclo simples em um grafo G, denotado por C_n, é um caminho simples $\{v_1, v_2, ..., v_n\}$ tal que $v_n v_1 \in E(G)$ e $n \geq 3$. Em um ciclo simples, o número de vértices é igual ao número de arestas e para todo vértice $v_i \in C_n$, $g(v_i) = 2$. Como exemplo, na Figura 2 temos um ciclo C_3 e um ciclo C_4.

O produto cartesiano $G_1 \times G_2$ de dois grafos G_1 e G_2, é o grafo que contém o conjunto de vértices $V(G_1 \times G_2) = V(G_1) \times V(G_2)$ e dois vértices $v = v_1, v_2$ e $u = u_1, u_2$ de $G_1 \times G_2$ são adjacentes se ou $[v_1 u_1 \in E(G_1)$ e $v_2 = u_2]$ ou $[v_2 u_2 \in E(G_2)$ e $v_1 = v_2]$.

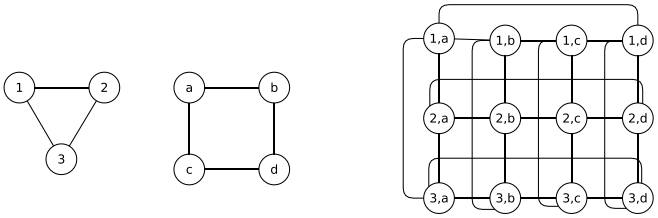


Figura 2. Exemplo de produto cartesiano dos grafos $C_3 \times C_4$.

Sejam G_1 e G_2 dois grafos, o *produto lexicográfico*, denotado por $G_1 \circ G_2$, é formado pelo conjunto de vértices de $V(G_1) \times V(G_2)$, sendo que os vértices $u = u_1, u_2$ e $v = v_1, v_2$ são adjacentes se $u_1 v_1 \in E(G_1)$ ou $u_1 = v_1$ e $u_2 v_2 \in E(G_2)$.

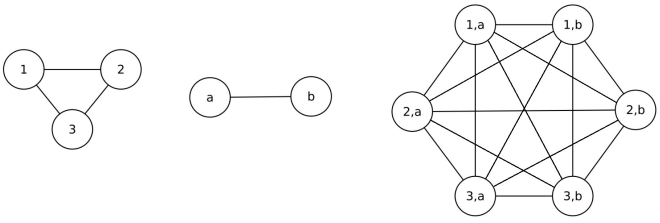


Figura 3. Exemplo de produto lexicográfico dos grafos $C_3 \times P_2$.

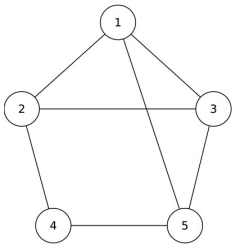
2. Algoritmos para caminhos

Em teoria dos grafos são estudados diversos problemas, e entre eles está o problema de caminhos em grafos. Existem inúmeros tipos de caminhos em grafos e dois deles são os caminhos mais longos e os caminhos mais curtos. Em uma definição breve, os caminhos mais longos em grafos não ponderados são caminhos entre pares de vértices que passam pela maior quantidade de vértices possíveis do grafo, e no caso de grafos ponderados são os caminhos entre pares de vértices onde a soma dos pesos é a maior possível. Os caminhos mais curtos em grafos não ponderados são os caminhos entre pares de vértices que passam pela menor quantidade possível de vértices, e no caso de grafos ponderados são os caminhos entre pares de vértices onde a soma dos pesos é a menor possível.

2.1. Matriz de adjacências

Uma *matriz de adjacências* é uma estrutura de dados do tipo matriz usada para armazenar um grafo a partir de suas adjacências. É usada para representar um grafo de maneira que os índices de linhas e colunas da matriz represente os vértices e a posição linha $\times$ coluna da matriz armazena se há ou não aresta incidente aos vértices. Assim, pode-se imaginar que uma matriz de adjacências tem um tamanho $n \times n$, onde n é o número de vértices. Porém, é facilmente detectado que a matriz é idêntica em suas posições acima e abaixo da *diagonal principal*, que é configurada pelas posições onde o índice da linha é igual ao índice da coluna, ou seja, a posição $(1, 2)$ da matriz é idêntica à posição $(2, 1)$. No caso de um grafo simples, verifica-se também que a diagonal principal tem todas suas posições iguais a zero, ou seja, a diagonal principal é nula. Logo, pode ser representada por uma matriz menor de tamanho $\frac{n\cdot(n-1)}{2}$ para grafos simples ou $\frac{n\cdot(n+1)}{2}$ para grafos que contêm laço, pois a diagonal principal não será nula.

Em grafos ponderados, a matriz armazena em suas posições o peso da aresta entre os vértices representados pelo índice da linha e da coluna, diferentemente da matriz para um grafo sem peso, que armazena apenas se há ou não uma aresta incidente aos vértices, como pode ser visto na Figura 4.



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	0
3	1	1	0	0	1
4	0	1	0	0	1
5	1	0	1	1	0

Figura 4. Exemplo de grafo simples e matriz de adjacências $n \times n$.

2.2. Algoritmo de Dijkstra

Um dos algoritmos usados para calcular o caminho mais curto em um grafo é o algoritmo de *Dijkstra*, que foi desenvolvido pelo matemático holandês Edsger Dijkstra, em 1959. O algoritmo é usado para encontrar o caminho mais curto entre todos os vértices de um grafo de arestas com pesos positivos, orientado ou não orientado. A grosso modo, o algoritmo calcula o menor caminho do vértice inicial até seus vértices adjacentes, e dos vértices

adjacentes até os próximos vértices sempre escolhendo a aresta de menor soma entre seu peso e sua distância até o vértice final do grafo.

De maneira mais formal, o algoritmo funciona da seguinte maneira em um grafo G com vértice inicial v_i, onde $d[a]$ é a distância até o vértice a e $\pi[a]$ é o vértice que antecede a. A *distância* armazena o peso calculado para ir até o vértice e o π armazena qual o vértice é seu antecessor no caminho. O algoritmo inicia atribuindo ao vértice inicial, chamado v_i, $d[v_i]$ = 0 e $\pi[v_i] = Nil$, e os demais vértices $v \in V(G)$ recebem $d[v] = \infty$, $\pi[v] = Nil$, até o seu vértice final v_f. Assim que estiver preparada a estrutura que irá armazenar o caminho mais curto, dá-se início ao processo de cálculo do caminho mais curto no grafo G representado por uma matriz de adjacências com pesos.

Algoritmo 1: *Dijkstra(G, s)*

Entrada: Um grafo ponderado G e um vértice $s \in V(G)$ inicial.
Saída: O caminho mais curto no grafo G.

```
1  para vértice v ∈ V(G) faça
2  |  d[v] ← ∞
3  |  π[v] ← NIL
4  fim
5  d[s] ← 0
6  π[s] ← NIL
7  S ← ∅
8  Q = V(G)
9  enquanto (Q ≠ ∅) faça
10 |  u ← EXTRACT_MIN(Q)
11 |  S ← S ∪ {u}
12 |  para vértice v ∈ Adj[u] faça
13 |  |  se (d[v] > d[u] + w(u, v)) então
14 |  |  |  d[v] ← d[u] + w(u, v)
15 |  |  |  π[v] ← u
16 |  |  fim
17 |  fim
18 fim
```

O Algoritmo 1 descrito é uma adaptação do algoritmo apresentado por [Cormen et al. 2009]. Nas linhas 1 a 6, o algoritmo inicializa a lista encadeada que armazenará o caminho mais curto através da distância representada por $d[v]$ e o vértice antecessor representado por $\pi[v]$.

A estrutura S utilizada na Linha 7 é uma lista encadeada que armazena os vértices do grafo que já passaram pelo algoritmo; sendo assim, inicia vazia e no fim do algoritmo terá todos os vértices do grafo armazenados. A estrutura Q é também uma lista encadeada que armazena os vértices do grafo que ainda não foram analisados, iniciada com todos os vértices do grafo e no fim fica vazia, pois o algoritmo testa todos os vértices do grafo.

Nas linhas 9 a 18 acontece a iteração que modifica o caminho durante o al-

goritmo, suas funções são: controlar as estruturas S e Q descritas através da função $EXTRACT\_MIN(Q)$ que retira e retorna o vértice de menor distância na estrutura passada por parâmetro e verificar se o caminho analisado no momento é menor que o caminho visto anteriormente. Caso seja, é feita a modificação nos campos de distância e vértice antecedente na estrutura que armazena o caminho mais curto. O método $w(u, v)$ calcula e retorna o peso das arestas incidentes aos vértices recebidos por parâmetro, no caso de uma matriz de adjacências retorna o valor da posição da aresta, o vértice u é utilizado como variável e $Adj[v]$ é o conjunto de vértices adjacentes ao vértice v passado por parâmetro.

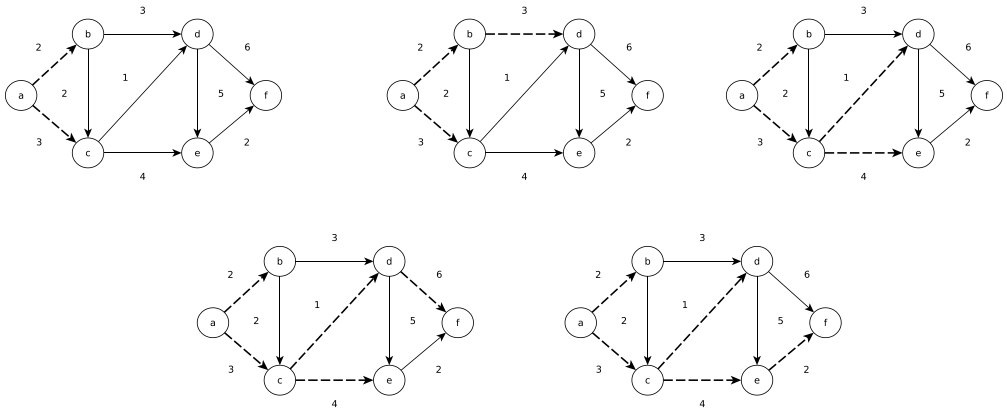


Figura 5. Exemplo de aplicação do algoritmo de Dijkstra.

Na Figura 5, temos uma ilustração do Algoritmo 1 passo a passo. Assim, pode-se perceber como é feita a escolha e a mudança das arestas que compõem o caminho em sua execução e, por fim, apresenta qual seria o caminho mais curto, representado por arestas pontilhadas, iniciando o algoritmo no vértice a do grafo.

Tabela 1. Estrutura que armazena o caminho mais curto do grafo na Figura 5.

Vértices	a	b	c	d	e	f
$d[v]$	0	2	3	4	7	9
$\pi[v]$	Nil	a	a	c	c	e

A Tabela 1 apresenta a estrutura de dados usada para armazenar os dados dos vértices no caminho. A estrutura Q descrita no Algoritmo 1 tem exatamente a configuração ilustrada e a função $EXTRACT\_MIN(Q)$ vista na Linha 10 do Algoritmo 1, retorna o vértice v que possui o maior $d[v]$ na estrutura Q, ilustrada na Tabela 1. Os valores apresentados na tabela correspondem aos valores do caminho calculado na Figura 5.

2.3. Caminho mais longo

Um grupo de turistas planejando a próxima viagem pelo Brasil, se reuniram para decidir quais cidades do país visitariam. Todos estavam de acordo que queriam visitar o maior número possível de cidades, com a única condição de não ter que repetir nenhuma. Coletaram todos os dados que necessitavam e após dias de estudo, alguns observaram que

era impossível percorrer todas as cidades sem repetir nenhuma e outros notaram que havia diversos percursos diferentes que passavam pelo maior número possível de cidades. A questão agora a ser observada é a mesma levantada por Gallai, Karger, Motwani e Ramkumar [Erdós and Katona 1966]. No exemplo dos turistas, se cada turista escolhesse um tal percurso distinto, teria alguma cidade que todos visitariam? No caso de Gallai, todo grafo conexo tem um vértice que aparece em todos os caminhos mais longos? E a questão respondida por Karger, Motwani e Ramkumar, [Erdós and Katona 1966], visto que é difícil traçar um caminho mais longo que passasse pelo maior número de cidades, será que ao menos poderiam encontrar um caminho mais longo onde visitariam uma fração do maior número possível de cidades?

Gallai [Erdós and Katona 1966] perguntou em 1966, em um colóquio em Tihany, se todo grafo conexo tem um vértice que aparece em todos os caminhos mais longos. A pergunta de Gallai é natural, pois é bem conhecido o fato de que em um grafo conexo, quaisquer dois caminhos mais longos sempre têm um vértice em comum. Porém, pouco tempo depois Walther [Zamfirescu 1976] deu uma resposta a essa questão construindo um grafo conexo provando que nem sempre é verdade que existe um vértice comum a todos os caminhos mais longos. O grafo de Walther está ilustrado na Figura 6, tem 25 vértices e 13 caminhos mais longos de tamanho 21 com intersecção vazia.

Com o surgimento do grafo de Walther, surgiu também a pergunta se o grafo de Walther era o menor possível. No começo dos anos 70, Walther e Voss [Walther and Voss 1974], e Zamfirescu [Zamfirescu 1976], responderam essa questão apresentando o grafo ilustrado na Figura 6 que tem 12 vértices, 9 caminhos de tamanho 9 (veja Tabela 2) que também responde negativamente à pergunta de Gallai.

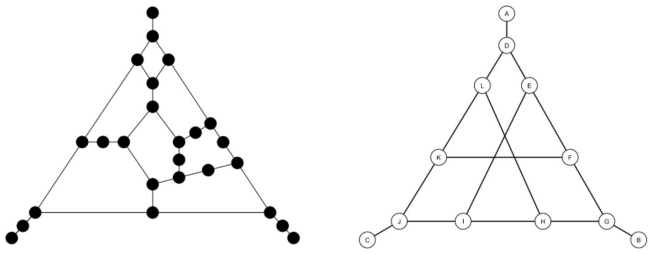


Figura 6. Grafo de Walther e grafo de Walther, Voss [Walther and Voss 1974] e Zamfirescu [Zamfirescu 1976].

2.3.1. Árvores

Uma árvore é um grafo conexo (existe caminho entre quaisquer de seus vértices) que não possui ciclos. Em uma árvore, cada vértice terá $1, ..., n$ vértices filhos e apenas um vértice pai, com exceção do vértice raiz da árvore que não possui vértice pai. Por volta de 1960, Dijkstra [Rezende 2014] propõe o primeiro algoritmo em tempo polinomial que encontra os caminhos mais longos em uma determinada classe de grafos, o algoritmo calcula os caminhos mais longos em uma árvore em tempo linear. Os Algoritmos 2, 3 e 4, formam uma adaptação do algoritmo de Dijkstra, utilizando para calcular o caminho mais longo, o algoritmo de busca em largura (*BFS*).

Tabela 2. Caminhos mais longos no grafo de Walther, Voss [Walther and Voss 1974] e Zamfirescu [Zamfirescu 1976].

Nº	Caminhos mais longos
01	$< A, D, E, I, H, G, F, K, J, C >$
02	$< A, D, E, I, H, L, K, F, G, B >$
03	$< A, D, E, I, J, K, L, H, G, B >$
04	$< A, D, L, H, G, F, E, I, J, C >$
05	$< A, D, L, H, I, E, F, K, J, C >$
06	$< A, D, L, H, I, J, K, F, G, B >$
07	$< B, G, F, K, L, D, E, I, J, C >$
08	$< B, G, H, L, D, E, F, K, J, C >$
09	$< B, G, H, L, K, F, E, I, J, C >$

Algoritmo 2: *BFS(G, s)*

Entrada: Um grafo árvore G e o vértice inicial s.
Saída: A árvore resultante da busca em largura de G.

```
1  para cada u ∈ V(G) − {s} faça
2  |  cor[u] ← branco
3  |  d[u] ← ∞
4  |  π[u] ← Nil
5  fim
6  cor[s] ← cinza
7  d[s] ← 0
8  π[s] ← Nil
9  Q ← s
10 enquanto Q ≠ ∅ faça
11 |  u ← pop(Q)
12 |  para cada v ∈ Adj[u] faça
13 |  |  se cor[v] = branco então
14 |  |  |  cor[v] ← cinza
15 |  |  |  d[v] ← d[u] + 1
16 |  |  |  π[v] ← u
17 |  |  |  push(Q, v)
18 |  |  fim
19 |  fim
20 |  cor[u] ← preto
21 fim
```

O Algoritmo 2 calcula a altura de todos os vértices de uma árvore G a partir da busca em largura, onde para todo $v \in V(G)$, $d[v]$ corresponde à sua altura na árvore resultante da busca em largura. O algoritmo é gerenciado por uma fila Q que armazena a sequência em que os vértices serão analisados.

As cores também são importantes para o processo, onde a cor branca significa um vértice ainda não descoberto pelo algoritmo, a cor cinza sinaliza um vértice que foi

descoberto, porém não finalizado e a cor preta um vértice já finalizado.

Algoritmo 3: *Muda_raiz(G, u*

Entrada: Um grafo árvore G e o vértice folha a ser raiz de G.
Saída: A árvore resultante da mudança de raiz.

```
1  v ← π[u]
2  direita[u] ← v
3  enquanto π[v] ≠ Nil faça
4  |  se direita[v] = Nil então
5  |  |  direita[v] ← π [v]
6  |  fim
7  |  senão
8  |  |  esquerda[v] ← π[v]
9  |  fim
10 |  v ← π [v]
11 fim
```

O Algoritmo 3 muda a raiz do grafo árvore G pelo vértice folha u, onde u é o vértice de maior altura na árvore resultante da busca em largura. Como u sempre será folha, logo o algoritmo não trata quando u tem filhos. O parâmetro $\pi[v]$ utilizado é resultante da busca em largura feita previamente.

Algoritmo 4: *CML_arvore(v)*

Entrada: Uma árvore G e raiz da árvore v
Saída: O caminho mais longo da árvore G.

```
1  BFS(G, v)
2  muda_raiz(G, u) - u é o vértice de maior distância resultante do
     BFS
3  BFS(G, u)
4  t é o vértice de maior distância, resultante do BFS
5  enquanto t ≠ Nil faça
6  |  F ← t
7  |  t ← π[t]
8  fim
9  retorna F
```

O Algoritmo 4 calcula o caminho mais longo na árvore G. O algoritmo faz a chamada dos algoritmos 2 e 3 vistos anteriormente e faz uso de uma fila F para armazenar o caminho mais longo. Os parâmetros $\pi[v]$ e distância utilizados são calculados no algoritmo de busca em largura. Na Figura 7 temos o exemplo de funcionamento do Algoritmo 4 para uma árvore com 8 vértices. O vértice pontilhado é o vértice calculado pelo Algoritmo 2 de maior altura e o caminho mais longo retornado pelo algoritmo está também representado por arestas pontilhadas.

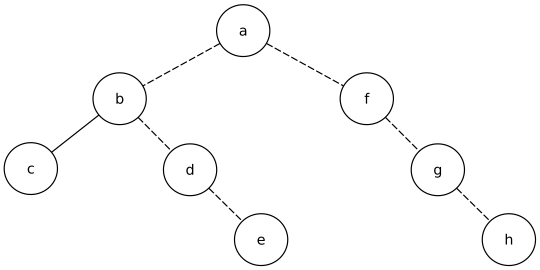


Figura 7. Exemplo de árvore e seu caminho mais longo.

2.3.2. Prisma complementar

Seja G um grafo simples e $\overline{G}$ seu complemento. O prisma complementar de G denotado por $G\overline{G}$ é o grafo formado a partir da união disjunta de G e $\overline{G}$, adicionando as arestas para um emparelhamento perfeito entre os vértices correspondentes de G e $\overline{G}$. O complemento $\overline{G}$ é o grafo cujo conjunto de vértices é $V(G)$ e cujas arestas são os pares de vértices não adjacentes de G. Considere n é o número de vértices de G, denotado por $|V(G)|$.

Proposição 1. *Se $G\overline{G}$ é um grafo prisma complementar, então $G\overline{G}$ tem $(\frac{n\cdot(n-1)}{2} + n)$ arestas.*

Demonstração. Seja $G\overline{G}$ um grafo prisma complementar. Pela definição de prisma complementar, temos que G é um grafo com n vértices e $\overline{G}$ seu complemento. Iremos provar por indução que o número de arestas de um grafo G e seu complemento $\overline{G}$ é $(\frac{n\cdot(n-1)}{2})$. Assumimos que G é um grafo com n vértices isolados. Temos, pela definição de complementar, que $\overline{G}$ será um grafo completo K_n com $(\frac{n\cdot(n-1)}{2})$ arestas. Seja m o número de arestas de G. Por hipótese, o complemento $\overline{G}$ terá $(\frac{n\cdot(n-1)}{2}) - m$ arestas. Portanto, temos que se o número de arestas de G é $m+1$, logo o número de arestas de $\overline{G}$ é $(\frac{n\cdot(n-1)}{2})-m-1$, pois a cada aresta e adicionada em G, e deixa de ser uma aresta de $\overline{G}$, iterando até que G passe a ser um K_n e $\overline{G}$ um grafo com n vértices isolados, com $(\frac{n\cdot(n-1)}{2})$ arestas. Pela definição de prisma complementar, para todo vértice $v \in G$ e $\overline{v} \in \overline{G}$, existe uma aresta $v\overline{v}$ adicionando assim n ao número de arestas de $G\overline{G}$. Logo, o número de arestas de um grafo prisma complementar $G\overline{G}$ é $(\frac{n\cdot(n-1)}{2} + n)$. □

Proposição 2. *Se $P_n\overline{P_n}$ é um grafo prisma complementar com $n > 5$, então o maior ciclo C_i em $P_n\overline{P_n}$ tem tamanho $i = 2n$ e $P_n\overline{P_n}$ é hamiltoniano.*

Demonstração. Seja P_n um caminho de u até v com $n > 5$. Pela definição de complemento, $\overline{P_n}$ é um grafo completo K_n, onde $P_n \not\subset K_n$. Assim, para cada vértice $\overline{t} \in \overline{P_n}$, com exceção de $\overline{u}$ e $\overline{v}$, $g(\overline{t}) = n - 3$ e $g(\overline{u}) = g(\overline{v}) = n - 2$. Logo, temos que $\overline{P_n}$ é um grafo de grau n e para todo vértice $\overline{s} \in \overline{P_n}$, $g(\overline{s}) \geq \frac{n}{2}$. Sendo assim, pelo teorema de Dirac [Dirac 1952], $\overline{P_n}$ é hamiltoniano, possuindo assim um ciclo hamiltoniano. Desse modo, existe um caminho hamiltoniano $P_i \in \overline{P_n}$ de $\overline{u}$ até $\overline{v}$. Pela definição de grafo prisma complementar para todo $v \in P_n$ e $\overline{v} \in \overline{P_n}$ a aresta $v\overline{v} \in P_n\overline{P_n}$. Assim temos que $P_n \cup v\overline{v} \cup P_i \cup u\overline{u}$ é um ciclo C_i que passa por todos os vértices de $P_n\overline{P_n}$. Como

$P_n\overline{P_n}$ tem $2n$ vértices, C_i tem $i = 2n$. Logo, C_{2n} é um ciclo hamiltoniano e $P_n\overline{P_n}$ é hamiltoniano. □

Algoritmo 5: *MC_ Prisma(G)*

Entrada: Um grafo $P_n\overline{P_n}$ dos vértices de a até n com $n > 5$.
Saída: O maior ciclo em $P_n\overline{P_n}$.

```
1  Insira os vértices em um vetor de tamanho n de ā até n̄
2  enquanto i ≠ n faça
3      u ← vet[i]
4      se (i + 1) = n então
5          F ← u; cor[u] ← preto
6          i ← 2; u ← vet[2]
7      fim
8      se i mod 2 ≠ 0 e (i + 2) = n então
9          F ← u; cor[u] ← preto
10         t ← vet[2]
11         v ← vet[n − 1]
12         F ← t
13         cor[t] ← preto
14         F ← v
15         cor[v] ← preto
16         i ← 4
17         u ← vet[4]
18     fim
19     t ← vet[i + 2]
20     se i mod 2 = 0 e cor[t] ≠ branco então
21         F ← u; cor[u] ← preto
22         v ← vet[n]
23         F ← v
24         cor[v] ← preto
25         i ← n
26     fim
27     senão
28         F ← u
29         cor[u] ← preto
30         i ← i + 2
31         se i = n então
32             v ← vet[n]
33             F ← v
34             cor[v] ← preto
35         fim
36     fim
37 fim
38 retorna P_n ∪ uū ∪ F ∪ vv̄
```

O Algoritmo 5 inicia inserindo todos os vértices de $\overline{P_n}$ em um vetor de vértices com tamanho n, iniciando do vértice $\overline{a}$ até $\overline{n}$. Este vetor é usado para gerenciar o andamento do algoritmo armazenando e atualizando as características dos vértices. A partir da Linha 2 temos a iteração que calcula o caminho entre $\overline{a}$ e $\overline{n}$ e todos os casos possíveis. A iteração é finalizada quando o caminho calculado chega a $\overline{n}$. Por fim, é feita a concatenação dos caminhos de P_n e $\overline{P_n}$, e das arestas de ligação $a\overline{a}$ e $n\overline{n}$.

Como visto, o Algoritmo 5 calcula o maior ciclo C_{2n} em grafos $P_n\overline{P_n}$, com $n > 5$. Temos pela proposição 2 que este C_{2n} é um ciclo hamiltoniano, pois passa por todos os vértices de $P_n\overline{P_n}$. Isto posto, é facilmente visto que se retirarmos qualquer aresta entre dois vértices u e $v \in C_{2n}$, tem-se deste modo um caminho mais longo entre u e v.

2.3.3. Grafos simples com pesos não negativos

Algoritmo 6: *Dijkstra_ 2(G, s).*

Entrada: Um grafo simples G com pesos não negativos.
Saída: Todos os caminhos mais longos do grafo G.

```
1  para vértice v ∈ V(G) faça
2  |  d[v] ← 0
3  |  π[v] ← NIL
4  |  cor[v] ← branco
5  fim
6  S ← ∅
7  Q = V(G)
8  enquanto (Q ≠ ∅) faça
9  |  u ← EXTRACT_MAX(Q)
10 |  S ← S ∪ {u}
11 |  para vértice v ∈ Adj[u] faça
12 |  |  se (d[v] > d[u] + w(u, v) e cor[v] ≠ preto) então
13 |  |  |  d[v] ← d[u] + w(u, v)
14 |  |  |  π[v] ← u
15 |  |  |  cor[v] ← cinza
16 |  |  fim
17 |  fim
18 |  cor[u] ← preto
19 fim
```

O Algoritmo 6 foi desenvolvido para calcular e listar todos os caminhos mais longos presentes em grafos simples com pesos não negativos. O algoritmo é uma adaptação do algoritmo de Dijkstra descrito anteriormente. A função $EXTRACT\_MAX(Q)$, vista na Linha 9 do Algoritmo 6, é uma função que retorna o índice na matriz de adjacências, do vértice de maior distância da lista encadeada Q. Por fim, são listados todos os caminhos que são armazenados na fila encadeada S ao fim de cada iteração.

Na Figura 8, temos um exemplo de grafo simples, não orientado e com arestas de peso não negativos. Na Tabela 3, temos os maiores caminhos do grafo ilustrado na figura.

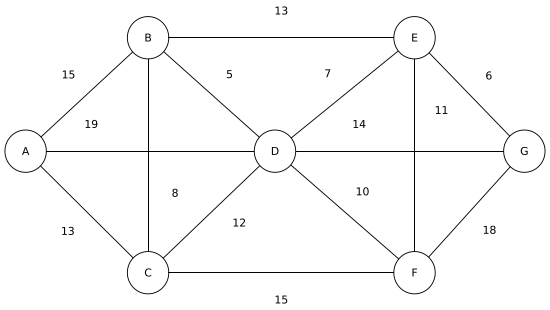


Figura 8. Exemplo de grafo simples com pesos não negativos.

Tabela 3. Caminhos mais longos presentes no grafo da Figura 8.

Nº	Caminhos mais longos
01	$< A, D, G, F, C, B, E >$
02	$< B, E, D, A, C, F, G >$
03	$< C, F, G, D, A, B, E >$

2.3.4. Grafos cordais formados por grafos completos K_n

Um grafo, G é dito *cordal* quando todo ciclo $C_i \in G$, com $i > 3$, tem pelo menos uma corda. Ou seja, se todo ciclo sem corda (csc) pertencente a G é um ciclo C_3, então G é um grafo cordal.

Seja G um grafo simples formado por um grafo completo K_n, e uma árvore de raiz v para todo vértice $v \in K_n$, diz-se que G é um grafo cordal K_n com n subárvores. Na Figura 9, temos um exemplo de grafo cordal e de um grafo cordal com 4 subárvores, formado por um K_4 com as subárvores de seus vértices e um de seus caminhos mais longos destacados pelas arestas pontilhadas.

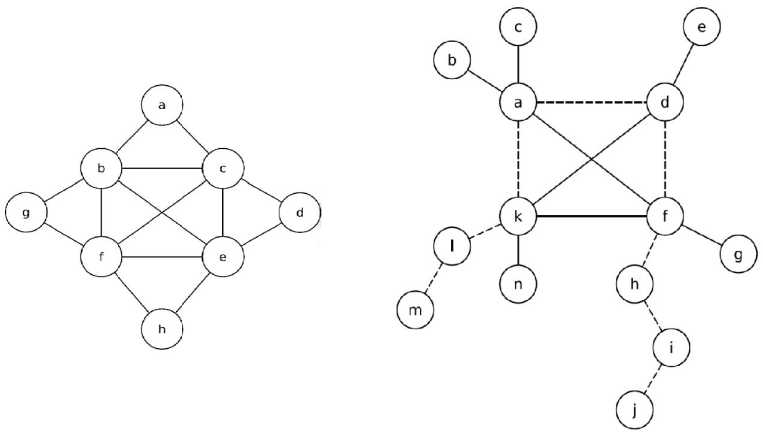


Figura 9. Exemplo de grafos cordais.

Algoritmo 7: *CML_Kn(G, u, v)*

Entrada: Um grafo K_n, o vértice inicial u e final v.
Saída: Um caminho mais longo em K_n de u até v.

```
1  Todo vértice t ∈ K_n inicia com cor branca.
2  t ← u
3  enquanto t ≠ v faça
4  |  se cor(Adj[t]) = branco e Adj[t] ≠ v então
5  |  |  F ← t; cor(t) ← preto
6  |  |  t ← Adj[t]
7  |  fim
8  |  senão
9  |  |  F ← t; cor(t) ← preto
10 |  |  F ← v; cor(v) ← preto
11 |  |  t ← v
12 |  fim
13 fim
14 retorna F
```

O Algoritmo 7 calcula o caminho mais longo entre dois vértices u e v em grafos completos K_n utilizando uma fila F para armazenar o caminho e a propriedade de grafos completos, que diz que há uma aresta entre quaisquer pares de vértices.

Algoritmo 8: *CML_Cordal(G)*

Entrada: Um grafo K_n com n subárvores.
Saída: Um caminho mais longo.

```
1  para todo u ∈ K_n faça
2  |  BFS(S_v, v)
3  fim
4  u - vértice em K_n de maior distância no BFS.
5  v - vértice em K_n de segundo maior distância no BFS.
6  F ← CML_arvore(S_u, u)
7  F ← CML_K_n(K_n, u, v)
8  F ← CML_arvore(S_v, v)
9  retorna F
```

O Algoritmo 8 usa o Algoritmo 2 em toda subárvore S_v iniciada pelos vértices $v \in K_n$. Desta forma, os vértices adjacentes a v que também pertencem a K_n não pertencem a S_v. Os vértices u e v nas linhas 3 e 4, são vértices de K_n onde o BFS calculou os maiores valores de d, localizando assim quais são as maiores subárvores do grafo. Feito isso, o algoritmo concatena os caminhos mais longos das subárvores e o caminho mais longo do grafo completo fazendo as chamadas dos algoritmos 4 e 7 vistos anteriormente.

3. CONCLUSÕES

No decorrer do trabalho, vimos que há uma grande dificuldade de encontrar os caminhos mais longos em grafos, dado que é um problema NP-completo. Esse fato dificulta o

desenvolvimento de algoritmos em tempo polinomial dado à complexidade do problema.

Visando determinar a dificuldade de problemas menores, os estudos foram iniciados com análises de classes mais específicas de grafos e desenvolvendo alguns algoritmos que nos auxiliaram a entender o comportamento dos caminhos mais longos nestas classes.

A princípio foram estudados, grafos ponderados e grafos prismas complementares onde conseguimos resultados teóricos sobre os tamanhos dos caminhos e o tamanho dos grafos e algoritmos que listam os caminhos mais longos. Em sequência, foram estudados os grafos árvores, grafos completos e grafos cordais cujos algoritmos calculam um caminho mais longo nos diferentes tipos de grafos.

Como trabalhos futuros, seria interessante alterar os algoritmos para que funcionem em tempo menor e encontrar resultados teóricos e algoritmos para outros produtos de grafos e outras classes de grafos, como grafos cordais e k-árvores.

Referências

Cormen, T. H., Leiserson, C. E., Rivest R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press, 3th edition.

Bondy, J. A. and Murty U. R. (2008). *Graph Theory, Graduate Texts in Mathematics*. Springer.

Dirac, G. A. (1952). *"Some theorems on abstract graphs."* Proceedings of the London Mathematical Society 3.1, pages 69–81.

Erdós, P. and Katona G. (1966). Theory of Graphs. Proceedings of the Colloquium held at Tihany, Hungrary. Academic Press, New York. Problem 4 (Gallai T.), page 362.

Rezende, S. F. (2014). "Caminhos mais longos em grafos". Dissertação de mestrado, USP.

Walther, H. and Voss H. J. (1974). *Über Kreise in Graphen*. VEB Deutscher Verlag der Wissenschaften, Berlin.

Zamfirescu, T. (1976). *On longest paths and circuits in graphs*. Mathematica Scandinavica, Vol. 38, pages 211–239.