Revista de Sistemas de Informação da FSMA
n. 24 (2019) pp. 55-66

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Intersecção de caminhos mais longos em algumas classes de grafos: resultados teóricos, implementação e execução de algoritmos

Gabriel Matheus Faria de Almeida, *Graduando em ciência da computação, INF-UFG*,
Elisângela Silva Dias, *Doutora em ciência da computação, INF-UFG*.

***Resumo*—Neste trabalho, foram estudados os caminhos mais longos e suas intersecções em produtos de grafos, grafos cordais, grafos ponderados, árvores e grafos prisma complementares. Foram obtidos resultados teóricos sobre os caminhos mais longos e suas intersecções e alguns algoritmos de tempo fatorial buscando entender o problema e o comportamento dos caminhos mais longos nos diferentes tipos de grafos. Por fim, é feita a análise da implementação dos algoritmos e dos testes feitos em laboratório, mostrando o tempo de execução e seu desempenho dado diferentes tamanhos de grafos, onde o leitor pode concluir a eficiência dos algoritmos e fornecendo os resultados obtidos nos testes para novas comparações com algoritmos futuros. Este trabalho é um dos resultados de uma iniciação científica realizada pelos autores.**

***Palavras-chave*—Caminhos mais longos, produtos de grafos, algoritmos, intersecção de caminhos.**

Intersection of longer paths in some graph classes: theoretical results, implementation and execution of algorithms.

***Abstract*—In this paper, we study the longest paths and their intersections in chordal graphs, weighted graphs, trees and complementary prism graphs. Theoretical results were obtained about the longest paths and their intersections and some factorial time algorithms in order to understand the problem and the behavior of the longest paths in the different graph types. Finally, we analyze the implementation of the algorithms and the tests made in the laboratory, showing the execution time and its performance given different graph sizes, where the reader can conclude the algorithms efficiency and providing the test results for new ones. comparisons with future algorithms. This work is one of the results of a scientific initiation by the authors.**

***Index Terms*—Longest paths, graph products, algorithms, intersection of paths.**

I. Introdução

Autor correspondente: Gabriel Matheus Faria de Almeida, *gabriel-matheus@inf.ufg.br*

UM grafo $G = (V(G), E(G))$ não orientado 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 é incidente a v_1 e v_2, e v_1, v_2 são *adjacentes* em G.

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. Na Figura 1a temos o exemplo de um laço, na Figura 1b temos um exemplo de arestas duplas e na Figura 1c temos um exemplo de grafo simples.

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. Assim, em um grafo orientado, o conjunto $E(G)$ passa a ser composto de pares ordenados de vértices, posto que o primeiro elemento designa o vértice de partida e o segundo elemento, o vértice de chegada. 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.

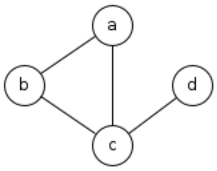
O *grau* de um vértice $v \in V(G)$, denotado por $g(v)$, é o número de arestas incidentes a ele. Por exemplo, na Figura 2, $\mathrm{g}(a) = 2$, $\mathrm{g}(d) = 4$ e $\mathrm{g}(e) = 3$.

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 $i\ j \in \{1, ..., n\}$ 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 $v_i \in C_n$,

(a) Exemplo de laço.

(b) Exemplo de arestas duplas.



(c) Exemplo de grafo simples.

Fig. 1: Exemplos de laço, arestas duplas e grafo simples.

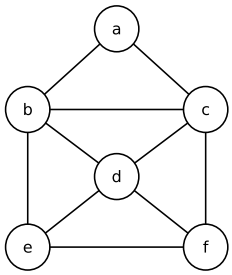


Fig. 2: Exemplo de grafo simples não orientado.

$g(v_i) = 2$. Como exemplo, na Figura 3a temos um ciclo C_3 e um caminho P_2 na Figura 3b.

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 $u_1v_1 \in E(G_1)$ ou $u_1 = v_1$ e $u_2v_2 \in E(G_2)$.

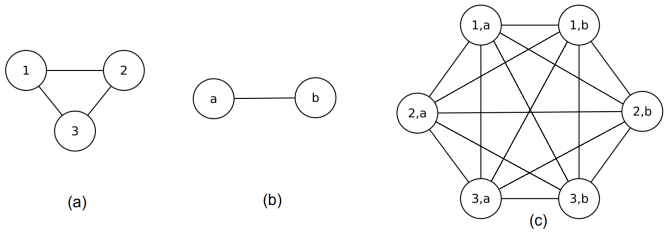


Fig. 3: Exemplo de produto lexicográfico dos grafos $C_3 \times P_2$.

Em um colóquio em Tihany, em 1966, Gallay [4] fez a pergunta que motivou o início do estudo de caminhos mais longos. Ele questionou se em todo grafo conexo, existia algum vértice que aparecia em todos os caminhos mais longos. Desta forma, iniciou-se o estudo baseado na busca de caminhos mais longos em grafos que respondiam negativamente a esta pergunta. Pouco tempo depois, Walther [7] mostrou a construção de um grafo que respondia negativamente a pergunta de Gallai, ou seja, não existia um vértice que se repetia em todos os caminhos mais longos. Desta forma, a busca de caminhos mais longos se mostrou um problema importante e de alta complexidade, onde listar os caminhos mais longos em um grafo é um problema NP-difícil [5], o que dificulta a análise da pergunta de Gallai.

Com a informação dos vértices que intersectam todos caminhos mais longos em um grafo tem-se os dados necessários para analisar vários problemas do mundo real. Como por exemplo, o local ideal para a instalação de um terminal rodoviário. Dado o conjunto de rotas de ônibus de uma cidade, o vértice de intersecção seria o ponto ideal para um terminal rodoviário, dado que este é o vértice "central" da topologia de rotas da cidade.

No decorrer deste trabalho, são estudados diversos tipos de grafos, analisando os problemas que cada classe traz e apresentadas as nossas contribuições para tentar resolver este problema apresentado por Gallay [4]. Na Seção II, é apresentado o problema de caminhos em grafos, assim como as formas de representação de um grafo e um algoritmo conhecido para o cálculo de caminhos. Na Subseção II-C é discutido mais pontualmente o problema de caminhos mais longos, dando uma pequena motivação, mostrando um caso real de caminhos mais longos e alguns grafos importantes. A partir daí, temos nos ítens II-C1 até II-C6 as classes de grafos que serão mais aprofundadas em nosso trabalho. Nelas são dadas definições e comportamentos mais aprofundados, mostrados alguns algoritmos que trouxeram a ideia de solução do nosso problema e nossas contribuições práticas e teóricas. Na Seção III é feita apresentação dos resultados experimentais obtidos e na Seção IV é concluido alguns resultados do trabalho e sugerido possíveis continuações do trabalho.

II. Algoritmos para caminhos

NA teoria dos grafos são estudados diversos problemas, e entre os quais inclui-se o problema de caminhos em grafos. Existem inúmeros tipos de caminhos entre os quais ressaltamos os caminhos mais longos e os 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.

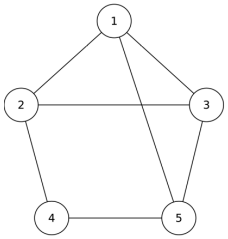
A. Matriz de adjacências

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 × coluna da matriz armazena se há ou não aresta incidente aos vértices. Assim, uma matriz de adjacências tem um tamanho $n \times n$, onde n é o número de vértices.

No caso de grafos não orientados, pode ser percebido 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 (i, j) da matriz é idêntica à posição (j, i), para todo par i, j. 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.

Dado que em nosso trabalho os grafos analisados são grafos simples, esta afirmação sobre a matriz de adjacências faz-se verdadeira para todo grafo. 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

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

B. 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. Resumidamente, 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 no menor caminho. O vetor *d[v]*, 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, um vértice $s \in V(G)$ inicial e peso *w(u,v)*
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 [2]. 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 algoritmo, 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

verifica se o caminho analisado no momento é menor que o caminho visto anteriormente. Em caso positivo, 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 e $Adj[v]$ é o conjunto de vértices adjacentes ao vértice v passado por parâmetro.

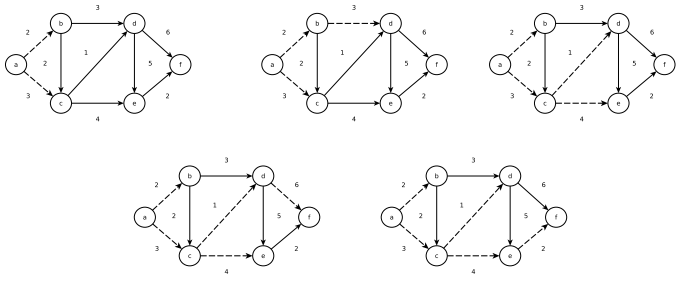


Fig. 5: Exemplo de aplicação do algoritmo de Dijkstra.

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 I: Representação da lista encadeada que armazena o caminho mais curto final do grafo apresentado 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 I 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 I. Os valores apresentados na tabela correspondem aos valores do caminho calculado na Figura 5.

C. 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 [4]. 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, [4], 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 [4] 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 [7] 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 de todos seus caminhos mais longos 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 [6], e Zamfirescu [7], responderam essa questão apresentando o grafo ilustrado na Figura 6 que tem 12 vértices e 9 caminhos de tamanho 9 (veja Tabela II) que também responde negativamente à pergunta de Gallai.

Com base nisto, em nosso trabalho, estudamos e discutimos diversas classes de grafos que introduzem ao leitor o problema de caminhos mais longos em grafos. As classes de grafos destacadas no decorrer do trabalho foram escolhidas e ordenadas através dos seus graus de complexidade ao tratar os caminhos mais longos. Com isso, conseguimos introduzir o tema de caminhos mais longos em grafos ligeiramente simples por início, mostrando as propriedades específicas de cada classe, nossas contribuições em cada uma delas e alguns resultados obtidos em nossa pesquisa.

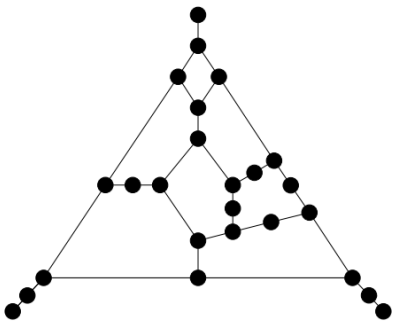


Fig. 6: Grafo de Walther [6] com 25 vértices.

1) Árvores: Uma árvore enraizada é um grafo conexo (existe caminho entre quaisquer de seus vértices) que não possui ciclos. Em uma árvore enraizada, 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 [5] propõe o primeiro algoritmo em tempo polinomial que encontra os caminhos mais longos em uma determinada classe de grafos, o algoritmo

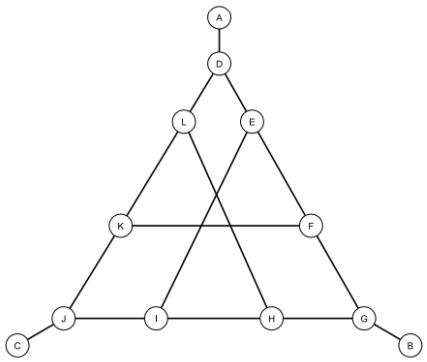


Fig. 7: Grafo de Walther, Voss [6] e Zamfirescu [7].

TABELA II: Caminhos mais longos no grafo de Walther, Voss [6] e Zamfirescu [7].

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	$< B, G, H, L, K, F, E, I, J, C >$

calcula os caminhos mais longos em uma árvore em tempo linear. Os Algoritmos 2, 3 e 4, formam uma adaptação do algoritmo de Dijkstra, onde nossa contribuição é a utilização do algoritmo de busca em largura (BFS) para calcular o caminho mais longo, dado que na literatura não há indícios de uma melhor ferramenta para tal, e o mesmo apresentou melhor tempo de execução em laboratório.

O Algoritmo 2 calcula a altura de todos os vértices de uma árvore G a partir da busca em largura em tempo $O(|V| + |E|)$, onde $|V|$ é o número de vértices da árvore e $|E|$ o número de arestas, 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.

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.

Essa mudança de raiz é importante para cálculo do caminho mais longo. Para melhor entendimento de sua justificativa, pode-se imaginar que o caminho mais longo está sendo calculado em duas etapas. Na primeira etapa, o algoritmo calcula o maior ramo (caminho) do lado esquerdo da raiz da árvore e escolhe o vértice mais distante da raiz. Em sua segunda etapa, o algoritmo modifica a raiz da árvore para este vértice mais distante calculado na primeira etapa, desta forma, a árvore fica com sua estrutura completamente desbalanceada, ou seja, que todas suas folhas estejam do lado direito da nova raiz da árvore. Por fim, é calculado o caminho mais longo a partir da nova raiz. Vale deixar claro, que esta idéia de mudança de raiz, é proposta por Dijkstra e citada em [5].

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 fim
```

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
```

Algoritmo 4: *CML_arvore(v)*

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

```
1  F ← ∅
2  u ← BFS(G, v)
3  muda_raiz(G, u) - u é o vértice de maior distância
     resultante do BFS
4  BFS(G, u)
5  t é o vértice de maior distância, resultante do BFS
6  enquanto t ≠ Nil faça
7  |  F ← t
8  |  t ← π[t]
9  fim
10 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 8 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.

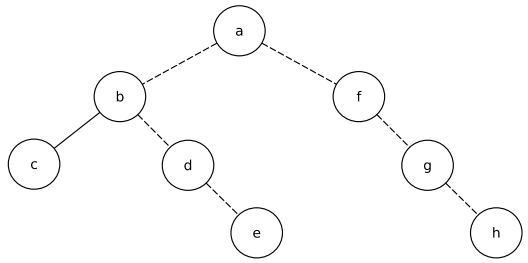


Fig. 8: Exemplo de funcionamento do Algoritmo 4.

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 $|V(G)|$. O grafo prisma complementar, é uma subclasse de grafos que pertencem à classe dos produtos lexicográficos, definidos na seção I. Na Figura 9 temos um exemplo de um grafo prisma complementar.

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

Prova: 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 $(\frac{n\cdot(n-1)}{2} + n)$. ■

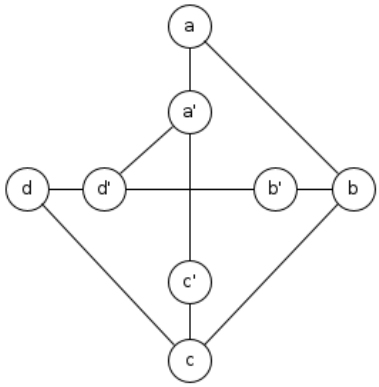


Fig. 9: Exemplo de um grafo prisma complementar.

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, portanto, $P_n\overline{P_n}$ é hamiltoniano.

Prova: 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 para todo vértice $\overline{s} \in \overline{P_n}$, $g(\overline{s}) \geq \frac{n}{2}$. Sendo assim, pelo teorema de Dirac [3], $\overline{P_n}$ 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. ■

Temos pela Proposição 2, que o grafo prisma complementar de caminhos $P_n\overline{P_n}$ com $n > 5$ é um grafo hamiltoniano. Portanto, é fácil notar que a intersecção de seus caminhos mais longos possui todos os seus vértices.

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

O Algoritmo 5 é usado para calcular o ciclo hamiltoniano definido e demonstrado na Proposição 2. Inicia-se 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 que possui todos os vértices de $P_n\overline{P_n}$. Isto posto, é possível visualizar 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. Nossa contribuição para a classe de grafos prisma complementar são as Proposições 2 e o Algoritmo 5.

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, consistindo em 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

TABELA III: Caminhos mais longos na Figura 10.

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*>

fim, são listados todos os caminhos que são armazenados na fila encadeada S ao fim de cada iteração.

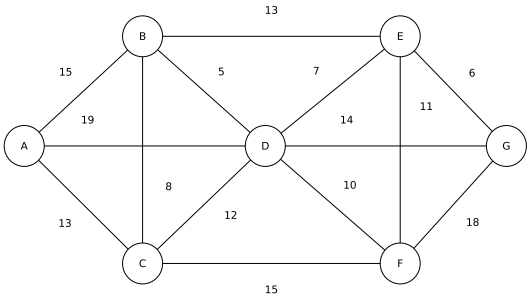


Fig. 10: Exemplo de grafo simples com pesos não negativos.

Na Figura 10, temos um exemplo de grafo simples, não orientado e com arestas de peso não negativos. Na Tabela III, temos os maiores caminhos do grafo ilustrado na figura.

4) Grafos k-árvores: Um grafo G é denominado uma k-árvore se G é um grafo completo de k vértices, ou se G é obtido a partir de uma k-árvore adicionando um vértice único de grau k. A classe de grafos 1-árvores é equivalente à classe das árvores. Pode-se observar na Figura 11 um exemplo de um grafo 3-árvore, onde parte-se de um grafo completo K_3 composto pelos vértices a, b e c, adicionando os vértices d e e de grau 3.

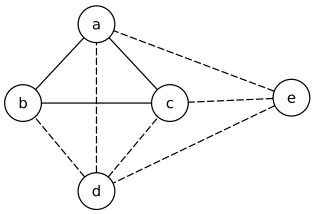


Fig. 11: Exemplo de um grafo 3-árvore.

Conforme a demonstração de Rezende [5], temos que para os grafos 2-árvores a resposta para a pergunta de Gallai é positiva, isto é, existe pelo menos um vértice comum a todos os caminhos mais longos nos grafos 2-árvores. Assim, conforme proposto pela própria autora, estudaremos a classe dos grafos k-árvores com $k \geq 3$.

5) 3-árvores cordais: A classe de grafos 3-árvores apresentada anteriormente tem sua definição de forma iterativa, onde em cada iteração é feita a adição de um novo vértice ao grafo 3-árvore anterior. Para melhor compreensão do problema, vamos definir a classe dos 3-árvores cordais.

Em um grafo 3-árvore a adição de um novo vértice e de suas incidências ocorre sem nenhuma restrição, ou seja, um novo vértice adicionado pode ser adjacente a quaisquer outros três vértices já existentes no grafo 3-árvore anterior. Dito isto, para definir a classe dos 3-árvores cordais, adicionamos uma restrição para este novo vértice.

Um grafo G é denominado 3-árvore cordal se G é um grafo completo K_3 de três vértices, ou se G é obtido a partir de uma 3-árvore cordal anterior, adicionando um vértice único v' de grau 3, onde v' é adjacente a v_1, v_2 e v_3 se existe um ciclo C_3 que contém v_1, v_2 e v_3. Na figura 12 temos a ilustração de um grafo 3-árvore cordal, onde foram adicionados os vértices d, e e f, nesta ordem. As arestas pontilhadas na Figura 12 representam as arestas adicionadas nas iterações da classe.

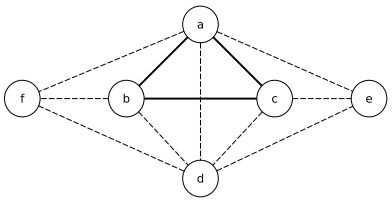


Fig. 12: Exemplo de um grafo 3-árvore cordal com a adição dos vértices d, e e f respectivamente, no grafo K_3 inicial.

A seguir, temos a demonstração de que a classe de grafos 3-árvores cordais pertence, de fato, à classe dos grafos cordais.

Proposição 3: Seja G um grafo 3-árvore cordal construído da forma descrita anteriormente, vamos mostrar que G pertence à classe dos grafos cordais.

Prova: Pela definição de 3-árvores cordais, o grafo inicial é um grafo K_3 que por definição pertence à classe dos grafos cordais e à classe dos 3-árvores. Logo, K_3 é um grafo 3-árvore cordal.

Seja G um grafo 3-árvore cordal, ao adicionarmos um novo vértice v' em G, onde v' é adjacente a v_1, v_2 e v_3 que pertencem a um ciclo C_3, temos que v' está presente em outros dois ciclos de tamanho 3, $C_3 = v_1$, v_2, v' e $C_3 = v_2$, v_3, v', onde a aresta v_1v_2 é uma corda no $C_4 = v_1$, v_2, v_3, v'. Assim, podemos dizer que $G = G \cup \{v'\}$ é um grafo que possui apenas ciclos C_3 sem cordas, pois para cada C_i em G com $i > 3$, existe pelo menos uma aresta e' incidente a dois vértices de C_i, onde e' não pertence às arestas de C_i. Sendo assim e' uma corda em C_i.

Portanto, como não existe C_i com $i > 3$ sem cordas em G, temos que G é um grafo cordal. Sendo assim, a classe de grafos 3-árvores cordais pertencem, de fato, à classe dos grafos cordais. ■

A definição da classe dos 3-árvores cordais é importante para a análise de seus caminhos mais longos, pois, nos grafos 3-árvores cordais é notável que a intersecção de vértices de todos seus caminhos mais longos são todos

os seus vértices. A seguir temos a demonstração desta afirmação.

Proposição 4: Seja G um grafo 3-árvore cordal construído da forma descrita anteriormente, vamos mostrar que todos os vértices de G pertencem aos caminhos mais longos de G e logo, a intersecção de seus caminhos mais longos são todos os vértices de G.

Prova: Inicialmente, temos que K_3 é um grafo 3-árvore cordal. Portanto sejam v_1, v_2 e v_3 os vértices de K_3, temos que o caminhos mais longo P de K_3 é o caminho $P = v_1, v_2, v_3$. Note que P contém todos os vértices de K_3.

Seja G um grafo 3-árvore cordal e P_g um caminho mais longo em G que contém todos os vértices de G, temos que ao adicionar um novo vértice v' em G, são possíveis três casos.

No primeiro caso, v' é adjacente ao vértice inicial v_i de P_g e outros dois vértices. Deste modo, é feito de forma direta a adição de v' ao caminho mais longo P_g, v' passa a ser o vértice inicial de P_g e portanto $v' \in P_g$ que é um caminho mais longo em $G \cup \{v'\}$.

No segundo caso, v' é adjacente ao vértice final v_f de P_g e outros dois vértices. Deste modo, ao adicionarmos v' ao caminho mais longo P_g de forma direta, v' passa a ser o vértice final de P_g e portanto $v' \in P_g$ que é um caminho mais logo em $G \cup \{v'\}$.

No terceiro e último caso, v' é adjacente a três vértices v_1, v_2 e v_3, que fazem parte de P_g, porém, nenhum deles é o vértice inicial ou vértice final de P_g. Suponha que existem as arestas v_1v_2 e v_2v_3 em P_g. Deste modo, temos por definição de 3-árvores cordais, que existe C_3 em G de tal forma que $C_3 = v_1$, v_2 e v_3. Como v' é adjacente a C_3 e todos os vértices de C_3 pertencem ao caminho mais longo P_g, basta adicionarmos o vértice v' ao corpo do caminho mais longo P_g. Dito isso, para que v' esteja em P_g, basta adicionarmos a aresta v_1v' e v_2v' em P_g. Note que outra forma de adicionar v' a P_g é adicionando a aresta v_2v' e $v3v'$. Logo $v' \in P_g$ que é um caminho mais longo em $G \cup \{v'\}$.

Por fim, podemos então concluir dos três casos, que um caminho mais longo em um grafo 3-árvore cordal passa por todos os seus vértices, e portanto a intersecção dos caminhos mais longos em um grafo 3-árvore cordal contém todos os seus vértices. ■

6) 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 (i.e., uma aresta entre dois vértices do ciclo que não faz parte do conjunto de arestas do ciclo, ver Figura 13). Ou seja, se todo ciclo sem corda (*csc*) pertencente a G é um ciclo C_3, então G é um grafo cordal.

Desta forma, usamos a definição de grafo cordal para definir uma subclasse dos grafos cordais, onde os grafos são formados por um grafo completo K_n e cada vértice $v \in K_n$, v é a raiz de uma nova árvore T. Note então, que cada grafo terá n árvores de diferentes tamanhos.

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-

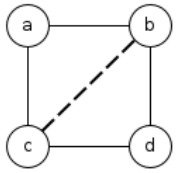


Fig. 13: Exemplo de corda no ciclo C_4. Onde a aresta destacada entre b e c não faz parte do ciclo, logo é uma corda.

se que G é um grafo cordal K_n com n subárvores. Na Figura 14, 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.

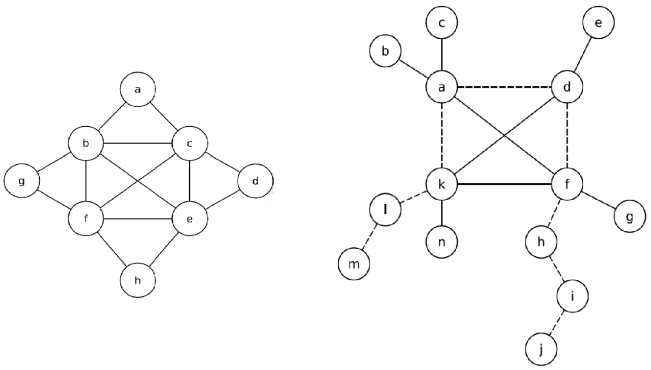


Fig. 14: Exemplo de grafos cordais.

Proposição 5: Seja G um grafo K_n cordal com n subárvores, então a intersecção de todos os seus caminhos mais longos possui todos os n vértices de K_n.

Prova: Seja G um grafo K_n cordal com n subárvores, suponha um conjunto A, onde para cada vértice $u \in A$, u é o vértices de subárvore com maior tamanho em G. Caso A tenha apenas um elemento de maior tamanho, faz-se necessário incluir os vértices de segundo maior tamanho em A, pois para o caminho mais longo em grafos K_n cordais com n subárvores, faz-se necessário pelo menos dois vértices das maiores subárvores em G.

Com isso, temos que os caminhos mais longos em G serão formados pelos vértices do conjunto descrito acima. Porém, temos que para todo par de vértices $u, v \in A$, u e $v \in K_n$, portanto, existe um caminho mais longo de tamanho n que passa por todos os vértices de K_n. Desta forma, para quaisquer pares $u, v \in A$, temos que os caminhos mais longos são formados pelas subárvores de u e v, e também pelo caminho mais longo de tamanho n de K_n que inicia em u e termina em v.

Assim, todos os caminhos mais longos de G possuem os n vértices de K_n.

Portanto, a intersecção dos caminhos mais longos de G tem tamanho n e possui todos os n vértices de K_n. ■

Algoritmo 7: *CML_Cordal(G)*

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

1 **para** *todo* $u \in K_n$ **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 $F \leftarrow CML\_arvore(S_u, u)$
7 $F \leftarrow CML\_K_n(K_n, u, v)$
8 $F \leftarrow CML\_arvore(S_v, v)$
9 **retorna** F

O Algoritmo 7 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 4 e 5, são vértices de K_n para os quais 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 a chamada do algoritmo 4 e do caminho mais longo do grafo K_n de tamanho n vistos anteriormente. CML_K_n é o algoritmo que retorna o caminho mais longo de K_n, que inicia em u e termina em v.

Nossas contribuições para a classe de grafos cordais formados por grafos completos K_n são: a Proposição 5 e o Algoritmo7.

A fim de ilustrar melhor a eficiência das nossas contribuições algorítmicas, foram realizados na Seção III testes experimentais da implementação dos algoritmos, onde mostramos resultados dos tempos de execução de cada algoritmo para diferentes tamanhos de grafos e fazemos uma análise mais detalhada das propriedades de implementação dos Algoritmos 5 visto nas seções anteriores e o Algoritmo 7.

III. Resultados experimentais

Dada a dificuldade de encontrar resultados experimentais e tempos de execução sobre a implementação de algoritmos de caminhos mais longos na literatura, esta seção é feita com o intuito de disponibilizar tais dados para comparação de algoritmos futuros. É válido ressaltar que não foi realizado nenhum estudo formal sobre a complexidade de tempo dos algoritmos apresentados, apenas os resultados experimentais extraídos dos testes executados em laboratório, que apresentam resultados satisfatórios dado o tempo de execução e o tamanho do grafo analisado. Assim, cabe ao leitor uma análise mais detalhada dos algoritmos para sua própria conclusão. Também é válido lembrar que encontrar caminhos mais longos em grafos, é um problema considerado NP-difícil e que os algoritmos apresentados neste trabalho são de tempo fatorial.

Na implementação do algoritmo 5 usamos uma estrutura de dados do tipo vetor, que armazena os vértices de $\overline{P_n}$. Essa estrutura permite a verificação das adjacências dos vértices de $\overline{P_n}$ em tempo constante e também é usada para auxiliar o cálculo do caminho entre os vértices de $\overline{P_n}$, onde é utilizado como parâmetro seus índices. Após ter calculado o caminho entre os vértices de $\overline{P_n}$ por uma iteração, o algoritmo concatena o caminho P_n com o caminho de $\overline{P_n}$ calculado e faz a união dos caminhos a partir das arestas $v\overline{v}$ da definição de grafo prisma complementar, retornando desta forma o ciclo C_{2n} com $n > 5$ analisado na Proposição 2.

A Figura 15 ilustra a execução do algoritmo em um grafo $Pn\overline{Pn}$ com $n = 6$. Onde a parte interna do grafo representa $\overline{Pn}$ e a parte externa representa o caminho Pn.

A implementação e execução do algoritmo foi realizada usando a linguagem C++ e o compilador g++ em um Sistema Operacional Linux Ubuntu versão 16.04.5 LTS, um Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz com 8GB de memória RAM e 500gb de disco rígido.

O tempo de execução (em milissegundos) para os grafos $Pn\overline{Pn}$, onde n varia de forma crescente de $[10, 500.000]$ estão representados na Tabela IV e ilustrado no gráfico da Figura 16, onde o eixo x representa a quantidade de vértices e o eixo y representa o tempo de execução do algoritmo.

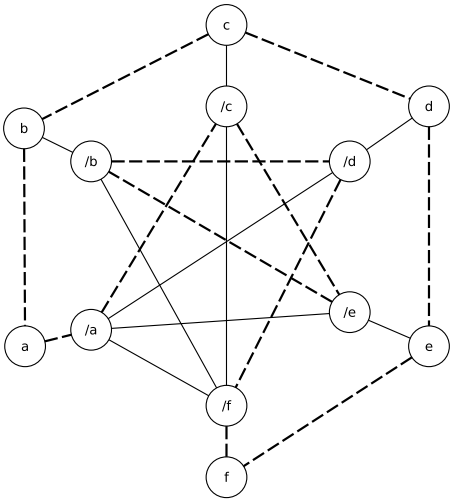


Fig. 15: Exemplo de execução do Algoritmo 5 em um grafo $P_6\overline{P_6}$.

Na implementação do algoritmo 7 usamos uma estrutura de dados do tipo matriz, onde cada uma de suas linhas armazenam o vetor obtido pela verticalização da parte triangular superior das matrizes de adjacências referentes ao grafo Kn e suas n subárvores, que são inseridas como entrada do programa. Desta forma, pode-se ter a busca das adjacências dos vértices de Kn cordal com n subárvores de forma estática e armazenando apenas os dados relevantes das matrizes inseridas na entrada do programa, que são usadas para execução do algoritmo. Não usaremos uma matriz $n \times n$ para armazenar cada parte do grafo, mas sim

TABELA IV: Tempo de execução do Algoritmo 5.

Nº vértices Pn	Tempo de execução (ms)
10	0,132828 ms
50	0,138433 ms
100	0,1469998 ms
500	0,1651832 ms
1.000	0,1829978 ms
10.000	0,6399396 ms
100.000	4,378896 ms
500.000	15,84056 ms

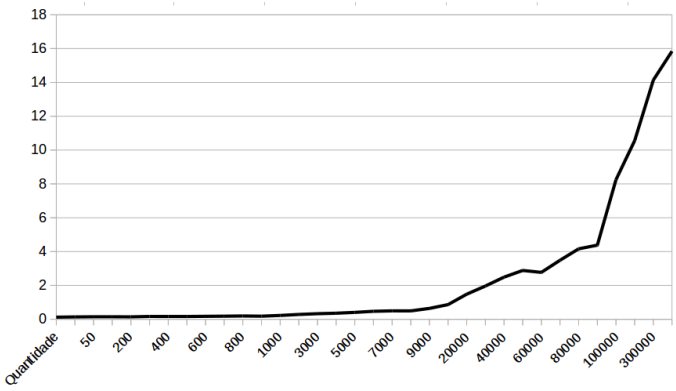


Fig. 16: Gráfico dos tempos apresentados pela Tabela IV, que são referentes ao Algoritmo 5.

um vetor de tamanho $\frac{n\cdot(n-1)}{2}$ para cada subárvore de Kn. Após ter calculado os caminhos das maiores subárvores de Kn com n subárvores, o programa insere a lista de caminhos mais longos a um arquivo de instância *.txt*, retornando então todos os caminhos mais longos de Kn com n subárvores.

O tempo de execução (em milissegundos) para os grafos Kn com n subárvores, onde n varia de forma crescente de $[5, 500]$ e o número de vértices de suas subárvores varia de $[10, 100]$ para cada n, estão representados na Tabela V e ilustrado no gráfico da Figura 17.

A implementação e execução do algoritmo foi realizada usando a linguagem C++ e o compilador g++ em um Sistema Operacional Linux Ubuntu versão 16.04.5 LTS, um Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz com 8GB de memória RAM e 500gb de disco rígido.

TABELA V: Tempo de execução do Algoritmo 7.

n/Nº vértices das n subárvores	Tempo de execução (ms)
5/10	1,717316 ms
5/100	24,59682 ms
10/10	4,731928 ms
10/100	41,9352 ms
30/10	7,017278 ms
30/100	132,0924 ms
50/10	10,234636 ms
50/100	225,4336 ms
100/100	464,336 ms

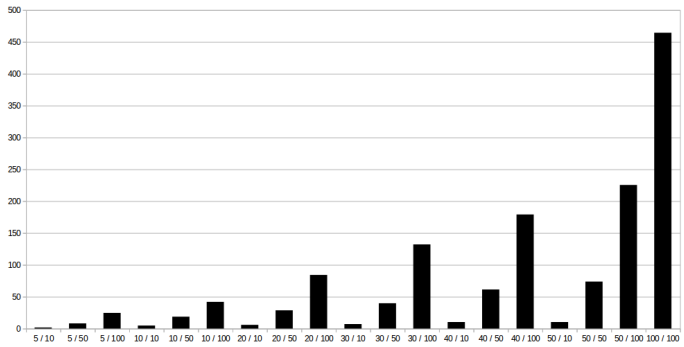


Fig. 17: Gráfico dos tempos apresentados pela Tabela V, que são referentes ao Algoritmo 7.

IV. Conclusões

No decorrer do trabalho, vimos que há uma grande dificuldade de encontrar os caminhos mais longos em grafos e definir a intersecção de vértices entre eles, dado que é um problema NP-difícil. 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 e verificar de que forma suas intersecções de vértices dos caminhos mais longos respondiam à pergunta de Gallai [4].

A princípio foram estudados, grafos ponderados e grafos prismas complementares onde conseguimos resultados teóricos sobre os tamanhos dos caminhos, tamanho dos grafos, resultados sobre suas intersecções de vértices dos caminhos mais longos e algoritmos que listam os caminhos mais longos. Em sequência, foram estudados os grafos árvores, grafos completos e grafos cordais, para os quais foram criados algoritmos que calculam os caminhos mais longos nos diferentes tipos de grafos.

Também foram apresentados resultados experimentais que auxiliam o leitor na análise do desempenho dos algoritmos implementados em laboratório. Dado a dificuldade de encontrar dados experimentais da implementação de algoritmos em caminhos mais longos, esta seção é fornecida para comparação em trabalhos futuros e auxílio na compreensão do trabalho realizado na iniciação científica da qual este artigo é fruto. É válido ressaltar a ausência de um estudo formal sobre a complexidade de tempo dos algoritmos apresentados na seção de experimentos, apenas os resultados experimentais extraídos dos testes executados em laboratório, que apresentam bons resultados dado o tempo de execução e o tamanho do grafo analisado.

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 k-árvores, grafos exoplanares e grafos cordais.

Referências

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

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

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

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

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

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

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