Teoria dos Grafos

Amanda Carolina Ambrizzi Ramin;
22.00721-0

Resumo. Este artigo explora a teoria dos grafos, abordando seus conceitos fundamentais e destacando sua relevância em problemas complexos de computação. São discutidos aspectos como a planaridade dos grafos e a implementação de algoritmos essenciais, como a busca em profundidade (DFS) para encontrar vértices, e os algoritmos de PRIM, Kruskal e Dijkstra para problemas de otimização. Também está incluso exemplos de aplicação afim de compreender a relevância dos grafos na resolução de problemas.

Palavras Chaves: Grafos, DFS, Dijkstra, PRIM, Kruskal

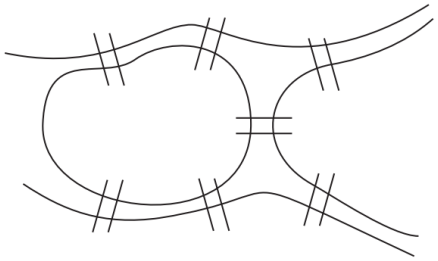
Abstract. This paper explores graph theory, covering its fundamental concepts and highlighting its relevance in complex computational problems. It discusses aspects such as graph planarity and the implementation of essential algorithms, including Depth-First Search (DFS) for finding vertices, and the Prim, Kruskal, and Dijkstra algorithms for optimization problems. Practical application examples are also included to demonstrate the importance of graphs in problem-solving.

Key Words: Graph, DFS, Dijkstra, PRIM, Kruskal

Introdução

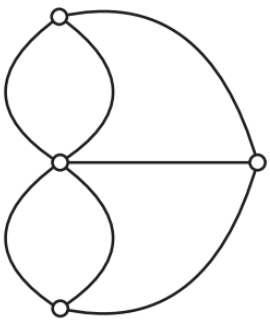
A Teoria dos Grafos surge no século XVIII na matemática, porém sua definição formal surge apenas no século XX. A primeira referência a grafo foi na cidade de Königsberg com o Problema das Pontes de Königsberg. A cidade era cortada pelo rio Pregel, que tinha duas ilhas. Para facilitar o transporte e passagem de pessoas foram construídas sete pontes nessa cidade como na figura 1.

Figura 1 – Mapa de Königsberg



Com o tempo, os habitantes da cidade passaram a se perguntar se havia uma forma de sair de um lugar e voltar para ele passando por todas as pontes sem repeti-las. Então, o matemático Leonhard Euler monta um diagrama da cidade associando cada cidade a um ponto (vértice) e cada ponte a uma reta (aresta), formando o que foi chamado de grafo representado na figura 2.

Figura 2 – Representação da cidade em forma de grafo



Euler percebe que há 2 pontos com uma quantidade ímpar de retas,

tornando impossível que uma pessoa saia de um desses pontos e retorne a ele sem repetir nenhuma reta. A ideia seria que é necessário um caminho para "sair" e um para "entrar", porém tendo três caminhos seria preciso repetir algum deles na volta. Um dos vértices com três arestas deve ser o início enquanto o outro é o final. E assim surgiu o primeiro grafo da história.

O conceito de grafo permitiu a modelagem de situações como redes de computadores, rotas de transporte, redes sociais, árvores genealógicas, química orgânica etc. Também pode ser associado a relações binárias como de parentesco.

Desenvolvimento

Definição de Grafo

Um grafo pode ser definido informalmente como um conjunto de vários pontos, chamados de vértices, e ligações entre esses pontos, chamados de arestas.

Formalmente, tem-se que um grafo simples G consiste em um conjunto finito e não vazio V(G) de objetos chamados vértices, juntamente com um conjunto E(G) de pares não ordenados de vértices, no qual seus elementos são chamados de arestas. Pode-se representá-lo por G = (V; E).

É importante definir mais alguns conceitos antes mostrar aplicações e implementações de algoritmo:

Arestas Paralelas: se duas ou mais arestas têm os mesmo vértices-extremidade, então essas arestas são paralelas.

Vértice Isolado: um vértice é isolado se não é extremidade de qualquer aresta.

Vértices Vizinhos: dois vértices unidos por uma aresta são chamados de adjacentes ou vizinhos.

Arestas Adjacentes: duas arestas são adjacentes se elas têm vértice comum.

Conjunto Vizinhança: denotado por N(v), é o conjunto de todos os vizinhos de um vértice fixo v.

Grau de um vértice: denotado por d(v), é o número de arestas que incidente em um vértice. Sendo assim, grau 0 é um vértice isolado e grau 1 é um vértice final.

Vértice par: quando o grau do vértice é par.

Vértice ímpar: quando o grau do vértice é ímpar.

Sequência de graus: são todos os graus do grafo escritos em ordem crescente podendo haver repetições.

Grafo simples: é um grafo que não tem loops nem arestas paralelas. O produto entre a quantidade de vértices e o grau deve ser par. Além disso, a soma dos graus de todos os vértices é igual ao dobro do número de aresta: $\sum_{v \in V} d(v) = 2\,|E|$ e a quantidade de vértices ímpares é sempre par.

Grafo completo: denotado por K_n , é um grafo que possui n vértices e tem $n\frac{n-1}{2}$ arestas.

Grafos Isomorfos: dois grafos são isomorfos se eles possuírem a mesma quantidade de vértices e arestas, além de quantidade iguais de vértices de um mesmo grau.

Subgrafo e Supergrafo: sejam dois grafos G1 e G2, diz-se que G2 é subgrafo de G1 se $V2 \subseteq V1$ e $E2 \subseteq E1$ e para toda arestas 'e' ∈ E2, se 'e' for incidente a v1 e v2, então v1, v2 ∈ V2. Dessa forma, também se diz que G1 é supergrafo de G2.

Grafo Bipartido: um grafo é dito como bipartido se seus vértices puderem ser separados em dois subconjuntos não vazios que não se interceptam, isso é, não deve haver arestas entre vértices do mesmo subconjunto.

Grafo Regular: é um grafo onde cada vértice tem o mesmo número de adjacência, ou seja, o mesmo grau. Um grafo regular com vértices de grau k é chamado de grafa k-regular.

Passeio: um passeio ocorre quando é possível chegar em um vértice a partir de outro seguindo uma sequência finita de vértices e arestas.

Grafo Conexo: um grafo é dito como conexo se existir um asseio entre dois quaisquer vértices.

Trilha: é um passeio em que não se repetem arestas. Toda trilha é um passeio, mas nem todo passeio é uma trilha. Uma trilha na qual o vértice final é igual ao inicial é chamado de circuito.

Caminho: um caminho é um passeio no qual não se repete vértices. Todo caminho é um passeio, mas nem todo passeio é um caminho. Um caminho no qual o vértice final é igual ao inicial é chamado de ciclo.

Grafo Euleriano: um grafo é dito como euleriano se possuir um passeio fechado que usa cada aresta do vértice uma única vez. Uma condição necessária e suficiente para o grafo conexo seja euleriano é que todos os vértices tenham grau par.

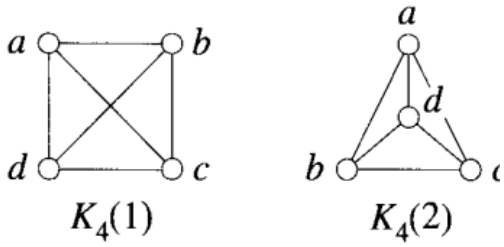
Grafo Hamiltoniano: um grafo é dito como hamiltoniano se existir um ciclo que passa por todos os vértices do grafo. Não há teoremas necessários para ser um hamiltoniano, mas há condições suficientes. Se o grafo tem n ≥ 3 vértices e todos os vértices têm grau pelo menos $\frac{n}{2}$, então o grafo é hamiltoniano (Teorema de Dirac). Se para cada par de vértices não adjacentes d(u) + d(v) ≥ n, então o grafo é hamiltoniano (Teorema de Ore).

Com isso, se encerra as definições básicas e suficientes para discussão dos próximos tópicos.

Conceito de Planaridade de Grafos

Um grafo G é dito planar se puder ser representado graficamente no plano de tal forma que não haja cruzamento de suas arestas. O número de cruzamentos é a quantidade de vezes que diferentes pares de arestas se cruzam, denotado por v(G). Ou seja, um grafo é planar se v(G) = 0, como no exemplo da figura 3.

Figura 3 – Duas possíveis representações de um grafo K_4

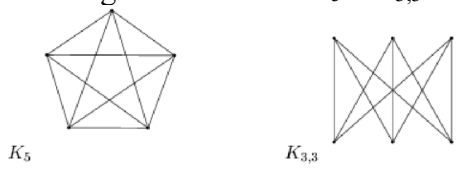


Agora, é fundamental definir o que são grafos homeomorfos. Um grafo homeomorfo é construído através da adição ou da eliminação de um vértice de grau 2. Se o grafo inicial for planar, então seus homeomorfos também serão. Além

disso, um grafo planar com mais de dois vértices tem no máximo 3v - 6 arestas.

Existem dois grafos importantes nessa teoria: K_5 e $K_{3,3}$, representados na figura 4.

Figura 4 – Grafos K_5 e $K_{3,3}$



Nota-se que há características comuns entre esse grafo: são regulares, não planares, a remoção de uma aresta ou um vértice torna o grafo planar, K5 não é planar com o menor número de vértices assim como K3,3 não é planar om o menor número de arestas.

Com isso, é apresentado o teorema de Kuratowski que enuncia: G é planar se e somente se G é homeomorfo a um grafo que não contém subgrafo homeomorfo a K_5 ou $K_{3,3}$.

Outro ponto é dizer que um grafo planar divide o plano em várias regiões chamadas de faces. Uma face é a área rodeada por arestas de um grafo sem contém nenhuma aresta.

As faces têm algumas propriedades:

Grau de uma face: é igual ao número de arestas contina da trilha que a define.

Número de faces: é sempre o mesmo independente da representação planar obtida.

Soma do grau das faces: a soma de todas as faces resulta em duas vezes a quantidade de arestas, já que cada arestas pertence a no máximo duas faces distintas.

A partir dessa ideia foi desenvolvida a Fórmula de Euler.

Fórmula de Euler: um grafo G tem v vértices, e arestas e f faces. Então a quantidade de faces do grafo é f = e - v + 2 faces.

Essa fórmula é dita como condição necessária, porém não é suficiente para determinar a planariadade de 100% dos casos. E então volta-se novamente a condição de Kuratowski que é necessário e suficiente para tal determinação.

Uma aplicação de planaridade seria em projeto de autoestrada ou de metrô, já que seria inconveniente ter várias linhas se cruzando, além de que ter menos cruzamentos possivelmente barateia a obra.

Implementações e Algoritmos

Após discutir outro conceito importante de grafo, é válido mostrar algumas implementações comuns aplicadas em grafos como a Busca em Profundidade, PRIM, Kruskal e Dijkstra.Todas as implementações estão em disponíveis em https://github.com/acramin/ED/tree/main/aula%2021%20-%20algoritmos%20de%20grafos/entrega/implementa%C3%A7%C3%B5es .

Implementação de Busca em Profundidade de Grafos

O objetivo é visitar todos os vértices e numerá-los na ordem em que são descobertos. O algoritmo funciona da seguinte forma:

1. Escolha um vértice inicial e comece a partir dele.

2. Marque o vértice como "visitado".
3. Para cada vértice adjacente não visitado:
    a. Chame o DFS recursivamente para explorar esse vértice.
4. Após explorar todos os vizinhos, retroceda e continue a explorar outros caminhos.

Esse funcionamento pode ser implementado por meio da uma pilha ou de forma recursiva. É aplicado em verificação de conectividade em redes, detecta ciclos em grafos direcionados e não direcionados.

Implementação do Algoritmo de PRIM

O algoritmo de PRIM tem a função de encontrar a árvore geradora mínima para um grafo conexo com pesos. Ou seja, constrói uma árvore que conecta todos os vértices com o menor peso possível. Esse algoritmo funciona da seguinte forma:

1. Escolha um vértice inicial e adicione-o à árvore geradora mínima.
2. Marque o vértice como visitado.
3. Encontre a aresta de menor custo que conecta um vértice já na árvore a outro ainda não visitado.
4. Adicione o vértice e a aresta a árvore geradora.
5. Repetir o processo até que todos os vértices estejam na árvore.

Esse algoritmo não funciona para grafos desconexos ou direcionais. É aplicado em sistemas de roteamento em redes de computadores, redes de distribuição elétrica.

Implementação do Algoritmo de Kruskal

O algoritmo de Kruskal também encontra a árvore mínima geradora de um grafo não direcionado com pesos. Em comparação com PRIM, esse algoritmo gera uma floresta geradora ao invés de uma única árvore. Seu funcionamento é da seguinte forma:

1. Coloque todas as arestas do grafo em uma lista e ordene-as em ordem crescente de peso.
2. Inicie a MST como conjunto vazio.
3. Para cada aresta em ordem de peso:
    a. Se a aresta conecta dois vértices em diferentes conjuntos, adicione-a à MST e junte os conjuntos dos vértices conectados.
4. Repita até que todos os vértices estejam conectados ou até que se tenha n-1 arestas.

Para fazer essa implementação é necessário usar Union-Find para verificar se dois vértices pertencem ao mesmo componente, para que dessa forma, se evite ciclos ao unir vértices. É aplicado em construção de redes com custo mínimo.

Implementação do Algoritmo de Dijkstra

O objetivo é caminho mais curto de um vértice de origem para todos os outros vértices em grafo com arestas de peso não negativo. Seu funcionamento é da seguinte forma:

1. Atribui custo inicial de zero ao vértice de origem e custo infinito para todos os outros vértices.
2. Usa uma fila de prioridade para selecionar o vértice de menor custo ainda não visitado e atualiza os custos dos vértices adjacentes.
3. Repete até que todos os vértices tenham sido visitados.

Esse algoritmo é aplicado principalmente em sistemas de navegação GPS para encontrar a menor rota possível.

Conclusão

Em suma, foi apresentado uma introdução à teoria dos grafos, desde sua origem histórica até os principais conceitos e algoritmos para resolver problemas complexos de computação. Discutiu-se também a importância da planaridade e sua aplicação em projetos como redes de transporte, onde minimizar interseções é essencial para reduzir custos e otimizar a construção.

Além disso, implementou-se algoritmos, como a busca em profundidade (DFS) para exploração de vértices, e os algoritmos de PRIM, Kruskal e Dijkstra para problemas de otimização, demonstrando a flexibilidade dos grafos na modelagem de problemas reais, como redes de comunicação, sistemas de navegação e distribuição de recursos.

Referências

FEOFILOFF, P. **Busca em profundidade em um grafo**. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_em_grafos/aulas/dfs.html>. Acesso em: nov. 2024.

FEOFILOFF, P. **Busca em profundidade (DFS)**. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html>. Acesso em: nov. 2024.

FEOFILOFF, P. **Algoritmo de Prim para árvore geradora barata num grafo**. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/prim.html>. Acesso em: nov. 2024.

FEOFILOFF, P. **Algoritmo de Kruskal para árvore geradora barata num grafo**. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html>. Acesso em: nov. 2024.

FEOFILOFF, P. **Algoritmo de Dijkstra para caminho barato num grafo com custos positivos**. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dijkstra.html>. Acesso em: nov. 2024.

HOLANDA, B. **Teoria dos GrafosOBM**. [s.l: s.n.]. Disponível em: <https://www.obm.org.br/content/uploads/2017/01/Nivel1_grafos_bruno.pdf>. Acesso em: nov. 2024.

JORGE, A. **Algoritmo de dijkstra: como calcular o caminho de custo mínimo?** Disponível em: <https://blog.betrybe.com/tecnologia/algoritmo-de-dijkstra/>. Acesso em: nov. 2024.

OLIVEIRA, V.; RANGEL, S.; DE ARAUJO, S. **Teoria dos Grafos**. [s.l:

s.n.]. Disponível em: <https://www.ibilce.unesp.br/Home/Departamentos/MatematicaAplicada/docentes/socorro/aula13_planar_2016_revsocorro.pdf>. Acesso em: nov. 2024.

PENNA, P. H. **Desenvolvendo Software | Algoritmo de Prim**. Disponível em: <http://desenvolvendosoftware.com.br/algoritmos/grafos/algoritmo-de-prim.html>. Acesso em: 5 nov. 2024.

SOARES DE MELO, G. **Introdução à Teoria dos GrafosRepositório Institucional da UFPB**. [s.l: s.n.]. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/tede/7549/5/arquivototal.pdf>. Acesso em: nov. 2024.