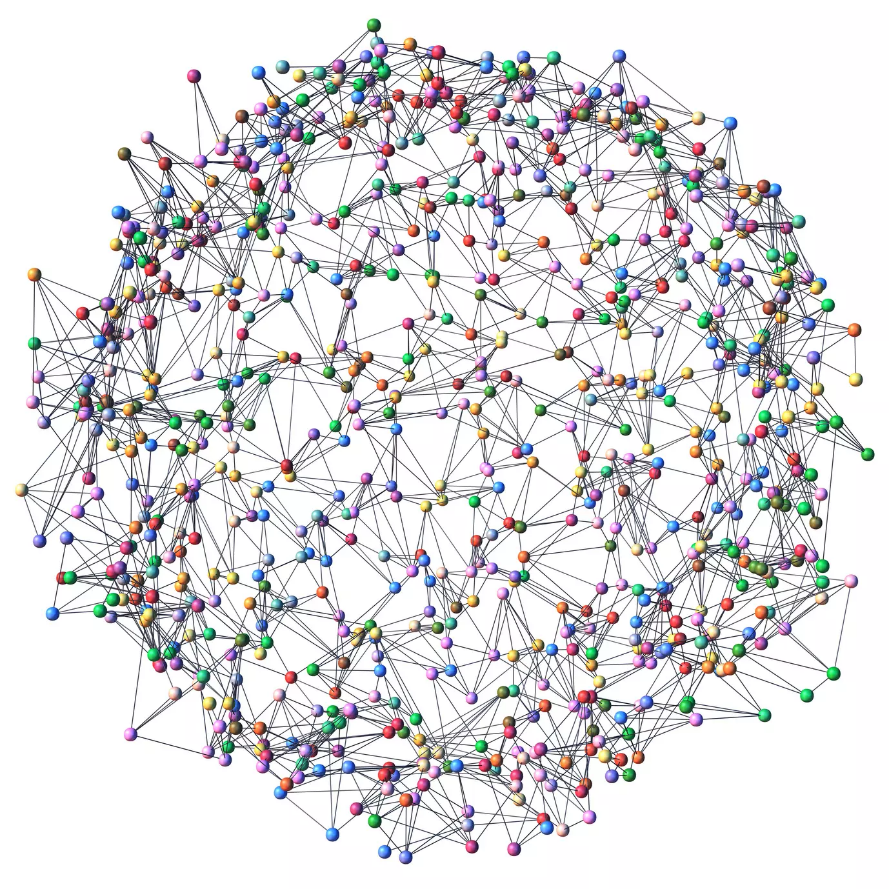


**Algoritmo de Prim**



**Licenciatura em Engenharia Informática**

**Desenho e Análise de Algoritmos - Projeto 3**

**2021-2022**

**Fabian Gobet - 97885**

**João Correia - 94576**

**Miguel Valadares - 98345**

Conteúdo

[Introdução 3](#_Toc103527681)

[Algoritmo de Prim 5](#_Toc103527682)

[Correção para o Algoritmo de Prim 8](#_Toc103527683)

[Implementações e Pseudocódigo 10](#_Toc103527684)

[Ordem Geral de Complexidade 12](#_Toc103527685)

[Melhor Caso 12](#_Toc103527686)

[Pior Caso 13](#_Toc103527687)

[Referências 14](#_Toc103527688)

# Introdução

Em teoria de grafos, considera-se uma *Spanning Tree* (ST) um subgrafo de um grafo G conexo, que contém todos os vértices do grafo G e é acíclico. Um grafo pode conter várias ST diferentes, no entanto todas estas possuem o mesmo número de vértices e arestas.

Num grafo G conexo e pesado podemos considerar o problema da ST cuja soma dos pesos das arestas utilizadas é o menor possível, isto é, Minimum Spanning Tree (MST).

O problema do cálculo da MST surge em várias áreas diferentes, nomeadamente no design de redes de telecomunicação, análise de agrupamento de dados e soluções aproximadas a problemas não determinístico-polinomiais. [1]

**Teorema 1.** Se é um grafo conexo e não orientado com vértices e uma ST de , então .

*Prova:* Por definição de ST vem que , de onde sai que .

**Teorema 2**. Se conexo e não orientado é uma ST então qualquer folha de tem grau 1.

*Prova:*  Suponhamos que existe uma folha com grau maior que 1 e seja o vértice dessa folha. Seja , vértices distintos e adjacentes a . Numa ST existe sempre um caminho entre qualquer par de vértices. Então existe um caminho da raiz, , para e também para . Como e são distintos, e também as suas arestas para Vi, então existe dois caminhos possiveis entre a e Vi, compondo assim um ciclo. Ora mas se existe um ciclo então não pode ser ST. Logo não é verdade que existe uma folha com grau superior a um.

**Teorema 3**. Se conexo e não orientado é uma ST, e uma aresta entre dois nós de , então remover resulta em duas ST e com .

*Prova:*  Suponhamos que , então existe um vértice que está em ambos os conjuntos, isto é, existe um caminho entre e . Mas então antes de remover existiam dois caminhos entre e , isto é G não seria ST por conter um ciclo. Logo segue que .

Por outro lado, se algum dos e não é ST então existe um ciclo em algum dos dois. Isto implica que não é ST pois por construção. Logo e são ST.

**Teorema 4.** Seja grafos conexos e não orientados tal que é uma ST de . Seja o conjunto das arestas de , então

*Prova:* Pelo teorema 1 sai de imediato que . Seja .

): Se e é ST de então não pode haver arestas em . Isto é

.

: Se e uma ST de então só é possível haver uma aresta em . Isto é .

Suponhamos que a propriedade é verdadeira para

Seja uma ST de um grafo tal que e sejam e vértices adjacentes de tal que é uma folha. Como é uma ST, existe um único caminho entre e .

Se retirarmos a aresta entre estes dois vértices, pelo teorema 3, ficamos com duas componentes desconexas, sendo cada uma destas uma ST para um subgrafo de T contendo apenas os respetivos vértices. Como é uma folha, pelo teorema 2 tem grau 1 no grafo G e grau 0 no respetivo subgrafo. Por outro lado, a ST do subgrafo resultante do qual não é vértice tem vértices e por hipótese .

Por construção vem que

# Uma imagem com homem, pessoa, fato, antigo Descrição gerada automaticamente Algoritmo de Prim

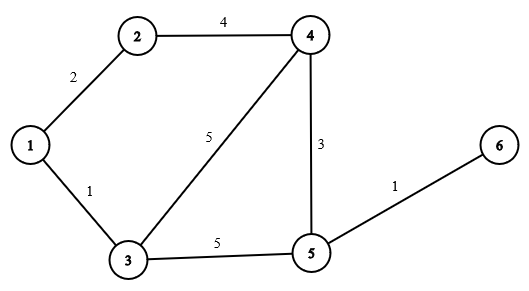
Em 1930, o matemático checo Vojtěch Jarník desenvolveu o primeiro algoritmo que resolvia o problema da MST, sendo mais tarde, em 1957, adaptado e republicado pelo matemático e cientista da área da computação Robert Clay Prim.

O algoritmo de Prim (também conhecido como o algoritmo de Jarník) consiste em encontrar uma MST num grafo orientado e pesado. [2]

Para tal, em primeira instância consideramos à priori pesos hipotéticos de cada aresta com um valor positivamente infinito, atribuindo um novo valor ao peso da aresta sempre que este seja menor que o último registado mediante a exploração de um nó relativo. De seguida escolhemos um nó aleatório do grafo G para inicializar o algoritmo, marcamos este como nó visitado e examinamos o peso de todas as suas arestas cujos nós adjacentes não tenham sido ainda visitados, identificado os mesmos e registando o peso associado a estes.

Depois, tendo em conta os pesos registados, escolhemos como próximo nó a explorar aquele que tem um menor peso associado, que ainda não foi explorado e já foi identificado. Doravante repete-se o processo de identificação de nós, atribuição de pesos e escolha do próximo nó a explorar até que todos os vértices do grafo sejam explorados. [3]

A título de exemplo segue-se uma coleção de imagens que mostram a ideia geral do algoritmo.



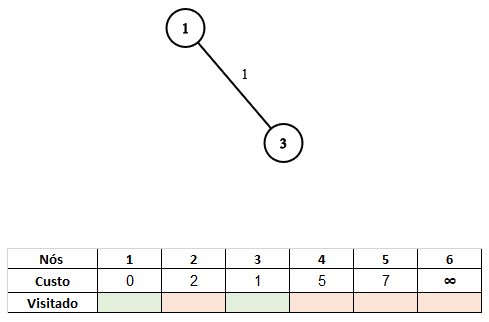
Consideremos o seguinte grafo G pesado e não orientado.

Estando nas condições de aplicar o algoritmo de Prim, escolhemos um vértice aleatório.

Uma imagem com mesa

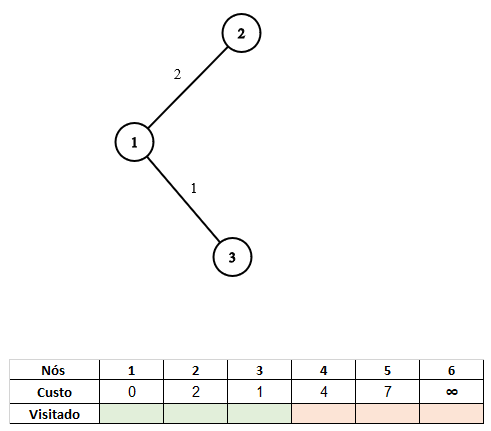
Descrição gerada automaticamente

Escolhendo o vértice 1 como inicializador do algoritmo, registamos este como visitado e atualizamos os valores dos seus adjacentes não visitados na tabela.



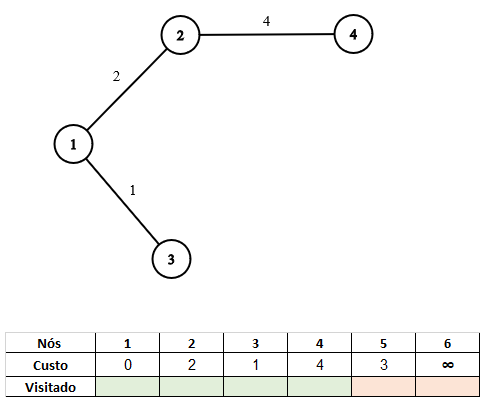
De seguida escolhemos como próximo vértice a explorar aquele que tem o menor valor na tabela e ainda não foi visitado. Neste caso será o 3.

Ao escolhermos o vértice, atualizamos o valor dos adjacentes na tabela

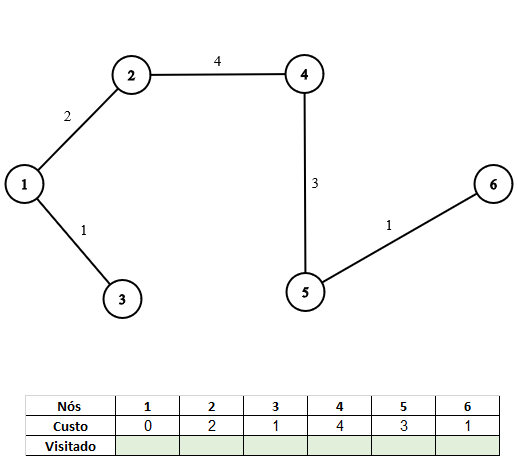


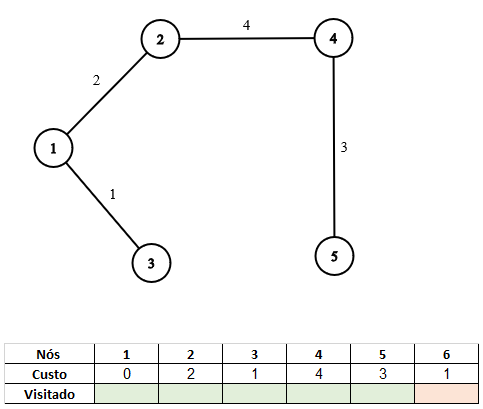
O próximo menor vértice não visitado era o 2, então escolhemos este como vértice a explorar, marcamos como visitado e atualizamos o valor dos adjacentes não visitados na tabela.

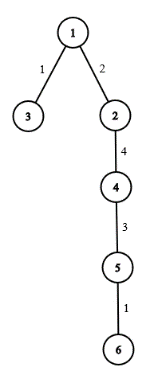
Vale a pena notar que no grafo original tanto 3 como dois são adjacentes a 4. Como 2 tem uma aresta com menos peso que a 3, atualizamos o respetivo valor.



Para os restantes passos segue recursivamente a mesma lógica até que todos os vértices tenham sido visitados.







No final a árvore resultante da aplicação do algoritmo de Prim ao grafo G deverá ser a que está representada na imagem à esquerda.

# Correção para o Algoritmo de Prim

A cada momento de decisão este algoritmo escolhe sempre o caminho cujo custo é o menor, sem reverter qualquer escolha feita anteriormente, remetendo assim a natureza deste algoritmo para uma estratégia do tipo greedy.

As vantagens de usar uma estratégia greedy são a facilidade de pensar e descrever o algoritmo, e implementar o mesmo. No entanto, algoritmos greedy nem sempre produzem casos ótimos, tornando-se assim fundamental analisar o algoritmo de Prim com vista a discriminar a optimalidade dos resultados produzidos por este.

**Teorema 5**. Seja um grafo conexo, não orientado com e uma ST de tal que , são vértices adjacentes pela aresta com , e a raiz de G. Seja = a primeira aresta no caminho entre e em , e o grafo que resulta de eliminar adicionar a . Então é uma ST de .

*Prova:* Suponhamos que não é ST, então existe um ciclo em . Pelo teorema 3 sabemos que as componentes isoladas que resultam de retirar a aresta são por si proprias ST e portanto não contêm ciclos. Ou seja, o ciclo estabeleceu-se ao introduzir . Mais ainda, esse ciclo tem de ser composto por dois caminhos diferentes entre as duas componentes.

Por construção está no máximo em um deles. Mas então também tem pelo menos dois caminhos entre as mesmas duas componentes, sendo um feito por , do qual concluímos que não é ST. Então não pode ser não é ST.

**Teorema 6 (Correção para o algoritmo de Prim)**. Seja um grafo conexo, não orientado e pesado uma ST de obtida através do algoritmo de Prim. Então é MST.

*Prova*: Suponhamos que G não é mínima com vista a chegar a uma contradição. Pelo teorema 4 existem Seja , , a sequência de arestas selecionadas, por ordem de escolha, pelo algoritmo de Prim.

Seja a aresta selecionada pelo algoritmo de Prim que condicionou G de forma a este já não ser mínimo doravante, onde e são vértices adjacentes ligados por , e seja U a MST de que contém o prefixo mais longo da sequência ES, isto é . Como é MST então existe um caminho entre os vértices e .

Seja a primeira aresta deste caminho e o conjunto de vértices da sequência tal que . Pelo teorema 3 vem que o grafo composto pelo conjunto de arestas é uma ST de .

Seja a função peso da aresta ou do grafo e consideremos os 3 casos possíveis:

Caso 1 - : Neste caso Q apenas difere de U por uma aresta que tem menos peso. Isto é, . Mas então entramos em contradição por ser uma MST.

Caso 2 - : Neste caso e, portanto, vem que Q também é uma MST. Mas se Q é uma MST e então não pode ter condicionado G no algoritmo de Prim e teria de ser que , o que é falso por construção.

Caso 3 - : Neste caso, como o peso da aresta é menor, no processo de aplicação do algoritmo de Prim devia ser escolhido em vez de , o que é falso por construção.

Como todos os casos são conducentes de uma contradição, concluímos que G é de facto MST.[4]

# Implementações e Pseudocódigo

O interesse em otimizar computacionalmente o algoritmo de Prim deu origem a várias implementações que diferem maioritariamente no tipo de estruturas utilizadas para definir uma aresta e para designar os próximos vértices a serem explorados.

As duas principais implementações denominadas, coloquialmente, por Classic Prim e PFS fundamentam-se nas estruturas de uma matriz de adjacência e de uma binary heap com indexação numa priority queue, respetivamente. A escolha entre estas duas opções deve ser feita tendo em conta a densidade do grafo sobre o qual o algoritmo vai ser aplicando, sendo favorável usar a implementação Classic Prim em grafos com uma densidade muito alta.

Para além das implementações e PFS, existem outras como a Fibonacci heap. No entanto, apesar de esta demostrar teoricamente melhores resultados, os respetivos testes empíricos não suportam essa conclusão. [5]

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Clássico | Binary heap | Fibonacci heap |
| Insert |  |  |  |
| Delmin |  |  |  |
| DecreaseKey |  |  |  |
| IsEmpty |  |  |  |
| Prim |  |  |  |

Capítulo 20, Algorithms in Java, 3ª Edição, Robert Sedgewick. [6]

Nesta revisão académica iremos considerar um pseudocódigo do algoritmo de Prim que tem como base uma binary heap indexada numa priority queue.

Uma imagem com texto

Descrição gerada automaticamente

# Ordem Geral de Complexidade

No pseudocódigo descrito anteriormente podemos identificar duas secções essenciais cuja sua completude determina o custo computacional associado a este.

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Numa análise casuísta podemos distinguir duas categorias que limitam o espetro de possibilidades: o melhor caso, o caso médio e o pior caso. Em todos estes três a primeira secção do algoritmo, a inicialização das estruturas e inserção na heap, mantêm-se com uma ordem de complexidade igual.

## Melhor Caso

O melhor caso acontece quando o grafo que a ser avaliado é por si próprio uma spanning tree (), com raiz no vértice escolhido para inicializar o algoritmo e todos os outros vértices como folhas, onde os vértices adjacentes são analisados imediatamente por ordem crescente de custo da aresta com a raiz.

Neste caso, na primeira secção a operação de insert na binary heap é constante, comprometendo um custo operacional na ordem de .

Uma imagem com texto

Descrição gerada automaticamente

Na segunda seccção será executada a operação de DecreaseKey do ciclo while uma única vez, dado que em todas as outras instâncias falha a aferição do if descriminado na imagem acima. Mais ainda, a operação de DecreaseKey não precisa de reordenar a binary heap, mudando apenas os valores na estrutura Key e sendo, portanto, constante. Posto isto, um custo computacional relativo à segunda secção na ordem de .

Segue-se que no melhor caso o algoritmo é da ordem de , sendo este também o limite inferior .

## Pior Caso

O pior caso acontece quando o grafo a ser avaliado é densamente máximo, o vértice que inicializa o algoritmo se encontra na última camada da heap e as arestas vão sendo descobertas por ordem decrescente.

Neste caso, na primeira secção a operação de insert na binary heap é constante, comprometendo um custo operacional na ordem de . No entanto, a operação DecreaseKey relativa ao vértice inicializador contribui com um custo de O(.

Na segunda secção será executada a operação de DecreaseKey do ciclo while tantas vezes quanto o número total Edges. Posto isto, concluimos um custo computacional relativo à segunda seccção na ordem de

Segue-se que no pior caso o algoritmo é da ordem de , sendo este também o limite superior .

Como o limite inferior é diferente do limite superior podemos apenas afirmar que o algoritmo é ordem geral de complexidade \*.

# Referências

|  |  |
| --- | --- |
| [1] | R. Sedgewick e K. Wayne, em *Algorithms*, Princeton University, Addison-Wesley, 1983, pp. 518-565. |
| [2] | “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Robert\_C.\_Prim. [Acedido em 13 Maio 2022]. |
| [3] | C. C. S. University, “Spanning Tree Prim's Algorithm - CCS University,” [Online]. Available: https://ccsuniversity.ac.in/bridge-library/pdf/MCA-Spanning-Tree-CODE-212.pdf. [Acedido em 10 Maio 2022]. |
| [4] | S. R. Tate, “Proof of Correctness for Prim's Algorithm - UNC Greensboro,” University of North Carolina at Greensboro, 15 Novembro 2016. [Online]. Available: https://home.uncg.edu/cmp/faculty/srtate/330.f16/primsproof.pdf. [Acedido em 11 Maio 2022]. |
| [5] | Computer Science Department at Princeton University, [Online]. Available: https://www.cs.princeton.edu/courses/archive/spr02/cs226/lectures/mst-4up.pdf. [Acedido em 14 Maio 2022]. |
| [6] | R. Sedgewick, Algorithms in Java, Princeton: Addison-Wesley Professional, 2004. |
| [7] | Y. Zhang, em *New Frontiers in Graph Theory*, IN-TECH, 2012. |
| [8] | R. Sedgewick e K. Wayne, “4.3 minimum spanning trees - Algorithms, 4th Edition,” [Online]. Available: https://algs4.cs.princeton.edu/lectures/keynote/43MinimumSpanningTrees-2x2.pdf. [Acedido em 14 Maio 2022]. |
| [9] | R. Muhamma, “Design and Analysis of Computer Algorithms,” Kent State University, [Online]. Available: https://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/primAlgor.htm. |