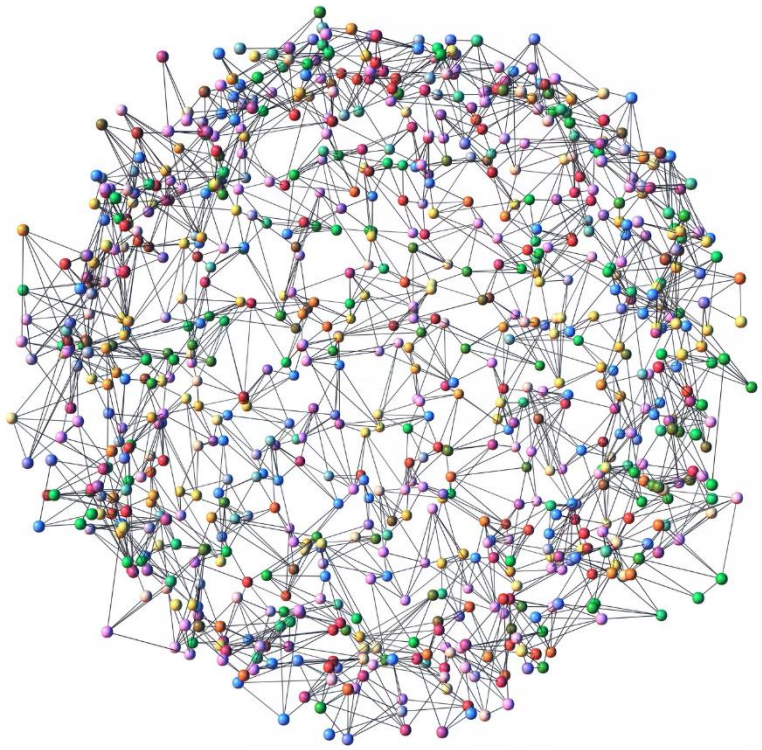
iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

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

FJM

Conteúdo

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 T é um grafo conexo e não orientado com $|V_T|$ vértices e G uma ST de T, então $|V_G| = |V_T|$.

Prova: Por definição de ST vem que $V_G = V_T$, de onde sai que $|V_G| = |V_T|$. ■

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

Prova: Suponhamos que existe uma folha com grau maior que 1 e seja V_i o vértice dessa folha. Seja V_j, V_k vértices distintos e adjacentes a V_i. Numa ST existe sempre um caminho entre qualquer par de vértices. Então existe um caminho da raiz, V_0, para V_k e também para V_j. Como V_j e V_k são distintos, e também as suas arestas para Vi, então existe dois caminhos possiveis entre a V_0 e Vi, compondo assim um ciclo. Ora mas se existe um ciclo então G não pode ser ST. Logo não é verdade que existe uma folha com grau superior a um. ■

Teorema 3. Se G conexo e não orientado é uma ST, e e_i uma aresta entre dois nós de G, então remover e_i resulta em duas ST G_1 e G_2 com $V_{G_1} \cap V_{G_2} = \emptyset$.

Prova: Suponhamos que $V_{G_1} \cap V_{G_2} \neq \emptyset$, então existe um vértice V_i que está em ambos os conjuntos, isto é, existe um caminho entre G_1 e G_2. Mas então antes de remover e_i existiam dois caminhos entre G_1 e G_2, isto é G não seria ST por conter um ciclo. Logo segue que $V_{G_1} \cap V_{G_2} = \emptyset$.

Por outro lado, se algum dos G_1 e G_2 não é ST então existe um ciclo em algum dos dois. Isto implica que G não é ST pois $G_1, G_2 \subset G$ por construção. Logo G_1 e G_2 são ST.■

Teorema 4. Seja T, G grafos conexos e não orientados tal que G é uma ST de T. Seja E_G o conjunto das arestas de G, então $|E_G| = |V_G| - 1 = |V_T| - 1$.

Prova: Pelo teorema 1 sai de imediato que $|V_G| = |V_T|$. Seja

$P(n)$: $O\ número\ de\ arestas\ no\ grafo\ G$ é $n - 1$.

$P(1)$: Se $|V_T| = 1$ e G é ST de T então não pode haver arestas em G. Isto é $0 = |E_G| = |V_G| - 1 = 0$.

$P(2)$: Se $|V_T| = 2$ e G uma ST de T então só é possível haver uma aresta em G. Isto é $1 = |E_G| = |V_T| - 1 = 1$.

Suponhamos que a propriedade é verdadeira para $P(n)$.

Seja G uma ST de um grafo T tal que $|V_T| = n + 1$ e sejam V_i e V_j vértices adjacentes de G tal que V_j é uma folha. Como G é uma ST, existe um único caminho entre V_i e V_j.

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 V_j 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 V_j não é vértice tem $\left|V_{G\setminus V_j}\right| = n$ vértices e por hipótese $|E_{G-V_j}| = n - 1$.

Por construção vem que $|E_G| = \left|E_{G\setminus V_j}\right| + \left|E_{V_j}\right| + 1 = n - 1 + 0 + 1 = n$. ■

Algoritmo de Prim

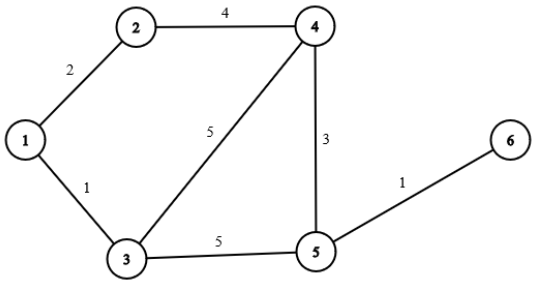
Em 1930, o matemático checo Vojtěch Jarník desenvolveu o primeiro algoritmo que resolvia o problema da MST, sendo mais tarde, em 1957, adaptado e republicado pelo matemático e cientista da área da computação Robert Clay Prim.

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.

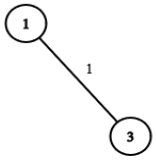


Consideremos o seguinte grafo G pesado e não orientado.

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

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

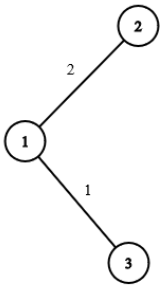
Nós	1	2	3	4	5	6
Custo	0	2	1	∞	∞	∞
Visitado						



Nós	1	2	3	4	5	6
Custo	0	2	1	5	7	∞
Visitado						

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.

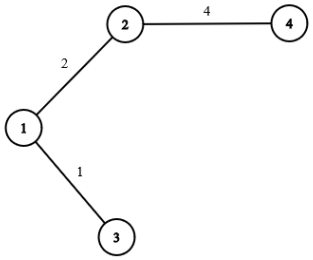
Ao escolhermos o vértice, atualizamos o valor dos adjacentes na tabela



Nós	1	2	3	4	5	6
Custo	0	2	1	4	7	∞
Visitado						

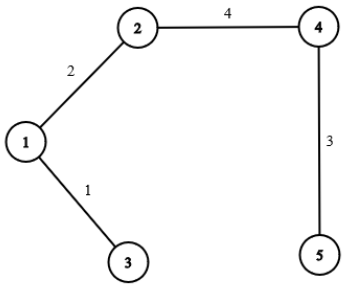
O próximo menor vértice não visitado era o 2, então escolhemos este como vértice a explorar, marcamos como visitado e atualizamos o valor dos adjacentes não visitados na tabela.

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.

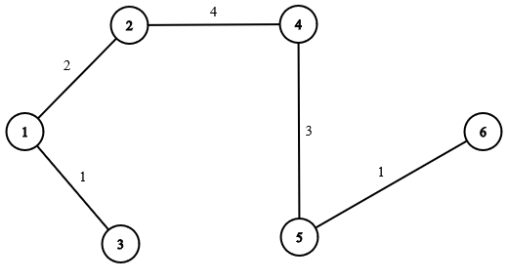


Nós	1	2	3	4	5	6
Custo	0	2	1	4	3	∞
Visitado						

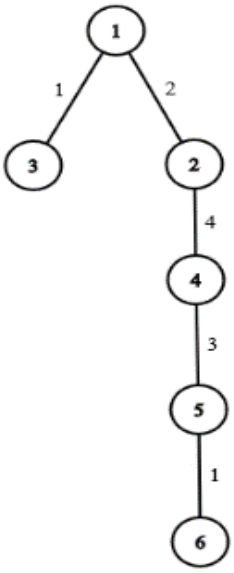
Para os restantes passos segue recursivamente a mesma lógica até que todos os vértices tenham sido visitados.



Nós	1	2	3	4	5	6
Custo	0	2	1	4	3	1
Visitado						



Nós	1	2	3	4	5	6
Custo	0	2	1	4	3	1
Visitado						



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 T um grafo conexo, não orientado com $|V_T| > 2$ e G uma ST de T tal que v_i, v_j são vértices adjacentes pela aresta e_j com $e_j \in T \wedge e_j \notin G$, e v_0 a raiz de G. Seja $e_i = \{\alpha, v_i\}$ a primeira aresta no caminho entre v_i e v_0 em G, e G' o grafo que resulta de eliminar e_i adicionar e_j a G. Então G' é uma ST de T.

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

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

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

Prova: Suponhamos que G não é mínima com vista a chegar a uma contradição. Pelo teorema 4 existem $n - 1$ $arestas\ em\ G$. Seja $ES = (e_1, e_1, \dots, e_i, \dots e_{n-1}), n = |V_T|$, a sequência de arestas selecionadas, por ordem de escolha, pelo algoritmo de Prim.

Seja $e_i = \{x, y\}$ a aresta selecionada pelo algoritmo de Prim que condicionou G de forma a este já não ser mínimo doravante, onde x e y são vértices adjacentes ligados por e_i, e seja U a MST de T que contém o prefixo mais longo da sequência ES, isto é $(e_1, e_1, \dots, e_{i-1})$. Como U é MST então existe um caminho entre os vértices x e y.

Seja $\{\alpha, \beta\}$ a primeira aresta deste caminho e V_W o conjunto de vértices da sequência $(e_1, e_1, \dots, e_{i-1})$ tal que $\alpha \in V_W$, $\beta \notin V_W$. Pelo teorema 3 vem que o grafo Q composto pelo conjunto de arestas $E_Q = (E_U \cup e_i) \setminus \{a, b\}$ é uma ST de T.

Seja $\omega(\{e_x, e_y\})$ a função peso da aresta ou do grafo e consideremos os 3 casos possíveis:

Caso 1 - $\omega(\{a, b\}) > \omega(\{x, y\})$: Neste caso Q apenas difere de U por uma aresta que tem menos peso. Isto é, $\omega(Q) < \omega(U)$. Mas então entramos em contradição por ser U uma MST.

Caso 2 - $\omega(\{a, b\}) = \omega(\{x, y\})$: Neste caso $\omega(Q) = \omega(U)$ e, portanto, vem que Q também é uma MST. Mas se Q é uma MST e $e_i \in Q$ então e_i não pode ter condicionado G no algoritmo de Prim e teria de ser que $e_i \in V_U$, o que é falso por construção.

Caso 3 - $\omega(\{a, b\}) < \omega(\{x, y\})$: Neste caso, como o peso da aresta $\{a, b\}$ é menor, no processo de aplicação do algoritmo de Prim devia ser escolhido $\{a, b\}$ em vez de $\{x, y\}$, 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	$\|V\|$	$O(\log_2 \|V\|)$	1
Delmin	$\|V\|$	$O(\log_2 \|V\|)$	$O(\log_2 \|V\|)$
DecreaseKey	1	$O(\log_2 \|V\|)$	1
IsEmpty	1	1	1
Prim	$O(\|V\|^2)$	$O(\|E\| \log_2 \|V\|)$	$O(\|E\| + \|V\| \log_2 \|V\|)$

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.

```
Prim(G,S):
    Input:
        - G: grafo conexo, pesado e não orientado, com n vértices
        - S: vértice aleatorio de G que inicializa o algoritmo
    Output:
        - MST

    Q <- inicializador da heap com elementos n elementos
    Key <- inicializador da estrutura dos pesos associados a cada nó
    MST <- inicializador da estrutura que guarda o pai associado a cada nó

    for v in [vértices de G]:
        Key(v) <- ∞
        MST(v) <- null
        Insert(Q,v)
    endfor

    DecreaseKey(Q,S,0)

    While (not(IsEmpty(Q)):
        U <- delmin(Q)
        for v in [vértices adjacentes a U]:
                if (v is in Q AND Key(v) > cost(U,v)):
                        MST(v) <- U
                        DecreaseKey(Q,v,cost(U,v))
                endif
        endfor
    endwhile

    return MST
endPrim
```

Ordem Geral de Complexidade

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

```
12      for v in [vértices de G]:
13          Key(v) <- ∞
14          MST(v) <- null
15          Insert(Q,v)
16      endfor
17
18      DecreaseKey(Q,S,0)
```

```
20      While (not(IsEmpty(Q)):
21          U <- delmin(Q)
22          for v in [vértices adjacentes a U]:
23                  if (v is in Q AND Key(v) > cost(U,v)):
24                          MST(v) <- U
25                          DecreaseKey(Q,v,cost(U,v))
26                  endif
27          endfor
28      endwhile
```

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 ($|E| = |V| - 1$), 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 $O(3|V|) = O(|V|)$.

```
if (v is in Q AND Key(v) > cost(U,v)):
        MST(v) <- U
        DecreaseKey(Q,v,cost(U,v))
endif
```

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 $O(3 * |V| + \sum_{1}^{|V|} \log_2 |V|) = O(\log_2(|V|!))$.

Segue-se que no melhor caso o algoritmo é da ordem de $O(\log_2(|V|!))$, sendo este também o limite inferior $\Omega(\log_2(|V|!))$.

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 $O(3|V|) = O(|V|)$. No entanto, a operação DecreaseKey relativa ao vértice inicializador contribui com um custo de $O(\log_2 |V|)$.

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 $O(|E| \log_2 |V|)$

Segue-se que no pior caso o algoritmo é da ordem de $O(|E| \log_2 |V|)$, sendo este também o limite superior $O(|E| \log_2 |V|)$.

Como o limite inferior é diferente do limite superior podemos apenas afirmar que o algoritmo é ordem geral de complexidade $O(|E|*\log_2 |V|)$.

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.