1 Árvores

Definição 1.1 *: Uma* ***árvore*** *é um grafo simples conexo e sem ciclos.*

Um grafo simples sem ciclos mas não conexo (em que cada componente conexa é portanto uma árvore) chama-se uma **floresta**.
Numa árvore (ou numa floresta), um vértice de grau 1 é uma **folha**. Uma árvore tem sempre pelo menos 2 folhas (considerar um caminho de comprimento máximo na árvore).

Ao contrário do que se passa para grafos em geral, o número de arestas de uma árvore é determinado pelo número de vértices: se T é uma árvore com n vértices então tem $n-1$ arestas. Provamos de facto um pouco mais:

Proposição 1.2 *: Seja G um grafo simples com n vértices. As seguites condições são equivalentes:*

a) G é uma árvore;

b) G é conexo e tem $n-1$ arestas;

c) G tem $n-1$ arestas e não tem ciclos.

Demonstração 1.3 *: Provamos $a) \Rightarrow b) \Rightarrow c) \Rightarrow a)$.*
Seja G conexo e sem ciclos; provamos por indução no número de vértices n que G tem $n-1$ arestas. Os casos $n=1$ e $n=2$ são evidentes; suponhamos portanto que a propriedade vale para grafos conexos, sem ciclos, com menos que n arestas; Se eliminarmos uma aresta uv de G ficamos com um grafo G' sem ciclos e com duas componentes conexas (se houvesse um caminho em G' de u para v, então em G esse caminho adicionado da aresta uv seria um ciclo). A hipótese de indução aplica-se a cada uma das componentes: se elas

têm, respectivamente, p e q vértices, terão $p-1$ e $q-1$ arestas; mas G tem os mesmos vértices e mais uma aresta que G', logo o número de arestas de G é $p-1+q-1+1=n-1$ como queríamos demonstrar.
Suponhamos agora que G é conexo e tem $n-1$ arestas; se G tivesse ciclos, poderíamos eliminar arestas mantendo o grafo conexo (para "abrir" os ciclos); no fim desse processo teríamos uma árvore com n vértices mas menos que $n-1$ arestas, o que já vimos ser impossível.
Suponhamos finalmente que G tem $n-1$ arestas e não tem ciclos. Sejam $G_1, \cdots G_k$ as componentes conexas de G e seja v_i o número vértices de G_i; cada G_i é uma árvore logo tem $v_i - 1$ arestas. Mas então

$$n-1 = \sum_{i=1}^{k}(v_i - 1) = \sum_{i=1}^{k} v_i - k = n - k$$

e portanto $k = 1$, ou seja, G é conexo.

Como consequência, a sequência de graus $d_1 \geq \cdots \geq d_n$ de uma árvore de ordem n satisfaz as condições

$$d_i > 0, \, \forall i; \qquad \sum_{i=1}^{n} d_i = 2(n-1)$$

Prova-se (por indução em n) que

Proposição 1.4 *: Dados inteiros positivos $d_1, \cdots, d_n$ existe uma árvore com vértices $v_1, \cdots, v_n$ e $d(v_i) = d_i$ se e só se*

$$\sum_{i=1}^{n} d_i = 2(n-1)$$

Nota 1.5 *Se a sequência de graus de um grafo simples G satisfaz*

$$\sum_{i=1}^{n} d_i = 2(n-1)$$

isso não implica que G seja uma árvore. No entanto, de acordo com a proposição provada atrás, se G for conexo aquela condição implica que se trata de uma árvore.

Outras caracterizações e propriedades de árvores deduzem-se facilmente e várias delas são deixadas como exercícios.

Exercícios 1

1. Dada uma árvore de ordem $n > 1$, mostrar que o número de folhas (vértices de grau 1) é igual a

   $$2 + \sum_{d(v)\geq 2} (d(v) - 2)$$

   em que a soma se faz sobre todos os vértices com grau maior ou igual a 2.

2. Demonstrar a Proposição **1.4**.

3. Seja T uma árvore com n vértices, tendo exactamente quatro deles grau 4, cinco grau 3, e todos os outros grau 1. Determinar o valor de n.

4. Sendo $m = \dfrac{\sum d(v)}{|V|}$ a média dos graus dos vértices da árvore $\mathbf{T}$, calcular o número de vértices $|V|$ em função de m.

5. Uma floresta F tem n vértices e m arestas. Quantas árvores (componentes conexas) tem F?

6. Mostrar que as seguintes condições são equivalentes:

   1) G é uma árvore;

2) para cada par de vértices u, v de G existe um único caminho em G de u para v.

3) G não tem ciclos mas se acrescentarmos uma aresta a, G tem exactamente um ciclo.

7. Mostrar que uma sequência

$$d_1 \geq d_2 \geq \cdots \geq d_n \geq 1$$

é a sequência ordenada de graus de uma árvore com $n > 1$ vértices $v_1, \cdots, v_n$, se e só se

$$\sum_{i=1}^{n} d_i = 2(n-1).$$

8. Dada uma árvore T com pelo menos 3 vértices, seja T' a árvore que se obtém eliminando todas as folhas de T. Mostrar que T e T' têm o mesmo centro.
   Concluir que o centro de T consiste em um único vértice ou em dois vértices adjacentes.

1.1 Árvores Geradoras

Definição 1.6 *:Uma **árvore geradora** de um grafo G é um subgrafo de G que contenha todos os vértices de G e seja uma árvore.*

Naturalmente, uma condição necessária para que G tenha uma árvore geradora é que seja conexo. Essa condição é também suficiente e existem algoritmos simples para a construção de uma árvore geradora.

1.1.1 Busca em largura e em profundidade

Descrevemos brevemente dois algoritmos de busca em grafos que podem ser usados para determinar uma árvore geradora. Ambos têm como ponto de partida a escolha de um vértice r (a raiz da árvore); em cada passo do algoritmo é actualizada uma lista de vértices visitados e há um vértice activo a partir do qual se continua a busca.

Busca em largura Na busca em largura a lista é iniciada com o vértice r e são adicionados sucessivamente ao fim da lista os vértices, não visitados anteriormente, adjacentes ao vértice activo, que é sempre o primeiro da lista; a cada novo vértice v adicionado à lista corresponde uma aresta da árvore incidente em v e no vértice activo; quando já não há novos vértices nessas condições, o primeiro vértice da lista é apagado e o vértice seguinte toma o lugar de vértice activo; o algoritmo termina quando a lista está vazia.

Busca em profundidade Na busca em profundiadade a diferença está em que o vértice activo em cada momento é o último da lista, que é apagado quando não é possível adicionar à lista um vértice adjacente ainda não visitado.

Em ambos os casos o algoritmo termina quando todos os vértices da componente conexa do grafo que contém o vértice r foram visitados e determina uma árvore geradora dessa componente conexa.

Exemplo 1.7 *Para exemplificar a aplicação destes algoritmos considere-se o grafo simpes com matriz de adjacência*

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Tomamos em ambos os casos o vértice 1 *como raiz e a busca é feita usando a ordem dos índices dos vértices; na busca em largura, a lista de vértices vai evoluindo do seguinte modo*

$$1 \to 1,2 \to 1,2,3 \to 1,2,3,4 \to 1,2,3,4,5 \to 2,3,4,5 \to 2,3,4,5,6 \to$$

$$\to 2,3,4,5,6,7 \to 3,4,5,6,7 \to 3,4,5,6,7,8 \to 3,4,5,6,7,8,9 \to$$

$$\to 4,5,6,7,8,9 \to 4,5,6,7,8,9,10 \to 5,6,7,8,9,10 \to$$

$$\to 5,6,7,8,9,10,11 \to 6,7,8,9,10,11 \to 6,7,8,9,10,11,12 \to$$

$$7,8,9,10,11,12 \to 8,9,10,11,12 \to 9,10,11,12 \to 9,10,11,12,13 \to$$

$$\to 10, 11, 12, 13 \to 11, 12, 13, \to 12, 13 \to 13 \to \emptyset$$

e a árvore tem as arestas

$$\{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{2,6\}, \{2,7\}, \{3,8\}, \{3,9\}, \{4,10\}, \{5,11\}, \{6,12\}, \{9,13\}$$

enquanto que na busca em profundidade temos

$$1 \to 1,2 \to 1,2,6 \to 1,2,6,8 \to 1,2,6,8,3 \to 1,2,6,8,3,4 \to$$
$$\to 1,2,6,8,3,4,10 \to 1,2,6,8,3,4,10,5 \to 1,2,6,8,3,4,10,5,11 \to$$
$$\to 1,2,6,8,3,4,10,5,11,9 \to 1,2,6,8,3,4,10,5,11,9,13 \to 1,2,6,8,3,4,10,5,11,9$$
$$\to 1,2,6,8,3,4,10,5,11 \to 1,2,6,8,3,4,10,5 \to 1,2,6,8,3,4,10$$
$$\to 1,2,6,8,3,4 \to 1,2,6,8,3 \to 1,2,6,8,3,7 \to 1,2,6,8,3,7,12$$
$$\to 1,2,6,8,3,7 \to 1,2,6,8,3 \to 1,2,6,8 \to 1,2,6 \to 1,2$$
$$\to 1 \to \emptyset$$

e a árvore tem as arestas

$$\{1,2\}, \{2,6\}, \{6,8\}, \{8,3\}, \{3,4\}, \{4,10\}, \{10,5\}, \{5,11\}, \{11,9\}, \{9,13\}, \{3,7\}, \{7,12\}$$

Estes algoritmos podem ser aplicados à solução doutros problemas como a determinação da distância entre dois vértices, a identificação dos vértices de corte do grafo, etc.

1.1.2 Contagem de árvores geradoras

Dado um grafo simples e conexo G qualquer, como determinar o número $t(G)$ das suas árvores geradoras? Para algumas famílias de grafos, muito em particular a dos grafos completos, como veremos mais adiante, é possível obter uma fórmula dependente apenas do número de vértices do grafo. No caso geral, uma das abordagens a este problema passa por encontrar uma relação de recorrência.

Se a é uma aresta de G, designamos por $G \setminus a$ o grafo que se obtém de G eliminando a; e por G/a o grafo que se obtém contraindo a, ou seja, eliminando a e identificando os dois vértices u e v incidentes em a num único vértice w. Note-se que G/a não será em geral um grafo simples, uma vez que se em G um vértice z é adjacente a u e a v, em G/a existem duas arestas paralelas entre z e w. Com estas notações, temos então o seguinte resultado:

Proposição 1.8 *: Se G é um grafo e $a \in A_G$,*

$$t(G) = t(G \setminus a) + t(G/a).$$

Demonstração 1.9 *: Se T é uma árvore geradora de G que contém a aresta a, então T/a é uma árvore geradora de G/a e vice-versa.*
Por outro lado, se T é uma árvore geradora de G que não contém a, é também árvore geradora de $G \setminus a$ e vive-versa.

Esta relação pode ser aplicada repetidamente até chegarmos a uma expressão de $t(G)$ como soma de parcelas fáceis de calcular directamente. Algumas observações:

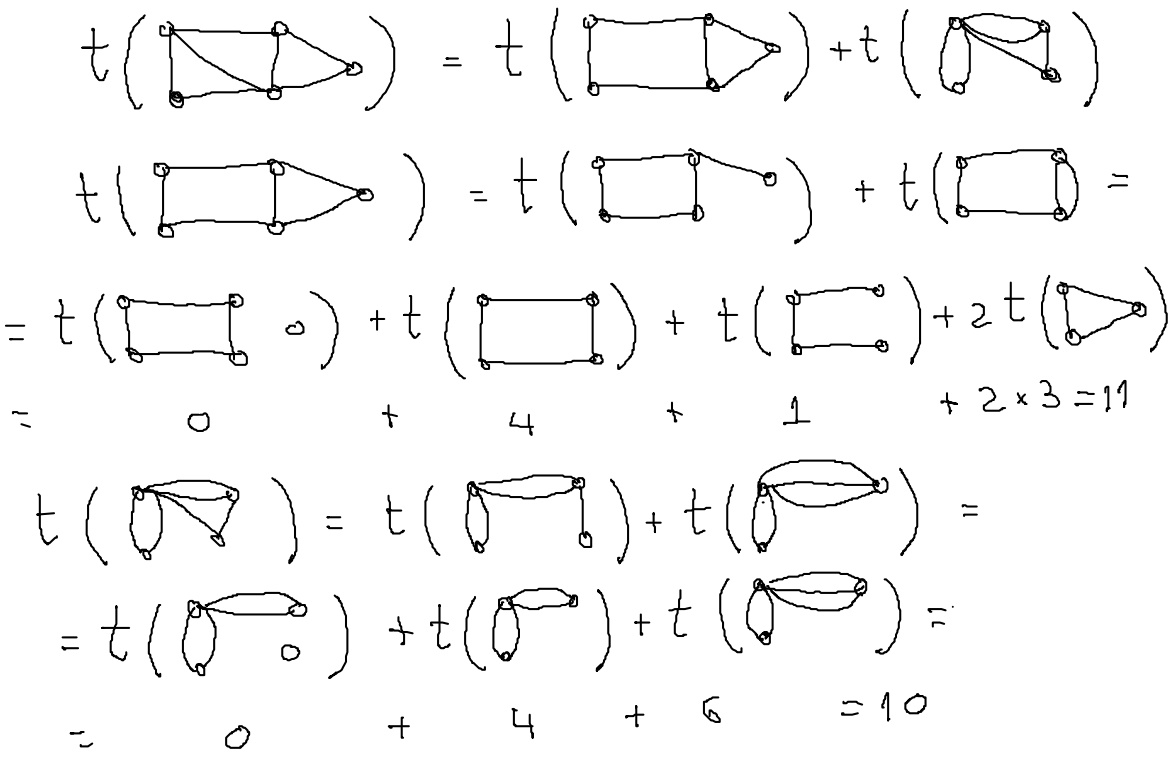
Se a for uma aresta de corte de G (isto é, se $G \setminus a$ for desconexo) a primeira parcela da soma é zero, o que corresponde ao facto de que qualquer árvore geradora de um grafo ter que conter todas as suas arestas de corte.

A relação de recorrência pode ser aplicada em qualquer dos sentidos, ou seja, quer considerando uma aresta a de G para eliminar/contrair, quer considerando uma aresta a que *não* está presente em G e acrescentando-a. No primeiro caso usamos a relação na forma $t(G) = t(G \setminus a) + t(G/a)$, e no segundo na forma $t(G \setminus a) = t(G) - t(G/a)$.

De acordo com o raciocínio feito na demonstração, não podemos ignorar as arestas paralelas que surgem no processo de eliminação/contracção de arestas, porque a escolha de cada uma dessas arestas paralelas para uma árvore geradora corresponde a diferentes árvores geradoras do grafo original; por outro lado, um lacete pode ser ignorado e eliminado porque nunca faz parte de uma árvore geradora.

Note-se ainda que se em alguma fase queremos eliminar/contrair arestas paralelas a, a' podemos fazê-lo simultaneamente, adaptando a fórmula: $t(G) = t(G \setminus \{a, a'\}) + 2t(G/\{a, a'\})$.

O exemplo seguinte ilustra a aplicação da fórmula, incluindo de algumas das observações anteriores:



9

Um outro problema de enumeração de árvores é o de calcular o número de árvores com vértices $v_1, v_2, \cdots, v_n$, ou seja, o número $t(K_n)$ de árvores geradoras do grafo completo K_n. A resposta é dada pela

Proposição 1.10 (Fórmula de Cayley)

$$t(K_n) = n^{n-2}$$

Uma das demonstrações desta fórmula foi descoberta por Prüfer e passa por atribuir a cada árvore um código (o código de Prüfer) $a_1 \cdots a_{n-2}$ onde $1 \leq a_i \leq n$.

A definição do código de Prüfer faz-se do seguinte modo: em cada passo, escolhemos a folha (vértice de grau 1) com menor índice, acrescentamos ao código o índice do vértice adjacente e eliminamos daárvore aquela folha e a respectiva aresta incidente; este procedimento é repetido até que tenham sido eliminados $n-2$ vértices e reste portanto uma árvore com dois vértices e uma aresta.
Note-se que o índice i ocorre no código $d(v_i) - 1$ vezes.

Exemplo 1.11 *Se T é a árvore com vértices $v_1, \cdots, v_{13}$ cuja matriz de adjacência é*

$$\begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

v_1 é a primeira folha e portanto a primeira entrada do código é 3*; a segunda é v_2 e a segunda entrada do código é de novo* 3*; tendo sido eliminadas as folhas anteriores, v_3 é agora uma folha o que implica que a próxima entrada do código é* 5*, e assim por diante. Conclui-se que o código de Prüfer de T é*

$$[3, 3, 5, 5, 5, 8, 8, 5, 12, 11, 12].$$

Esta definição estabelece uma bijecção entre as árvores de vértices $v_1, \cdots, v_n$ e o conjunto

$$\{[c_1, \cdots, c_{n-2}] : 1 \leq c_i \leq n\}$$

que tem obviamente n^{n-2} elementos.
Para ver que assim é, vamos determinar directamente a aplicação inversa, que a cada sequência faz corresponder uma árvore que tem exactamente essa sequência como código: dada a sequência

$$[c_1, \cdots, c_{n-2}],$$

iniciamos a construção da árvore com duas listas vazias V (vértices) e E (arestas); em cada passo identificamos o primeiro vértice $v \notin V$ e cujo índice não está na sequência; juntamos v a V, (v, v_c) a E, onde c é a primeira entrada da sequência, e apagamos c da sequência.
Quando a sequência está toda apagada restam dois vértices por escolher que são unidos por uma aresta.

Exemplo 1.12 *: seja a sequência*

$$[2, 5, 1, 1, 10, 7, 2, 3, 3, 5, 3];$$

vamos indicar em cada linha da tabela seguinte a lista V, a aresta criada e o

estado da sequência depois de cada passo:

$\{4\}$	$\{v_4, v_2\}$	$[5, 1, 1, 10, 7, 2, 3, 3, 5, 3]$
$\{4, 6\}$	$\{v_6, v_5\}$	$[1, 1, 10, 7, 2, 3, 3, 5, 3]$
$\{4, 6, 8\}$	$\{v_8, v_1\}$	$[1, 10, 7, 2, 3, 3, 5, 3]$
$\{4, 6, 8, 9\}$	$\{v_9, v_1\}$	$[10, 7, 2, 3, 3, 5, 3]$
$\{4, 6, 8, 9, 1\}$	$\{v_1, v_{10}\}$	$[7, 2, 3, 3, 5, 3]$
$\{4, 6, 8, 9, 1, 10\}$	$\{v_{10}, v_7\}$	$[2, 3, 3, 5, 3]$
$\{4, 6, 8, 9, 1, 10, 7\}$	$\{v_7, v_2\}$	$[3, 3, 5, 3]$
$\{4, 6, 8, 9, 1, 10, 7, 2\}$	$\{v_2, v_3\}$	$[3, 5, 3]$
$\{4, 6, 8, 9, 1, 10, 7, 2, 11\}$	$\{v_{11}, v_3\}$	$[5, 3]$
$\{4, 6, 8, 9, 1, 10, 7, 2, 11, 12\}$	$\{v_{12}, v_5\}$	$[3]$
$\{4, 6, 8, 9, 1, 10, 7, 2, 11, 12, 5\}$	$\{v_5, v_3\}$	$[]$

Neste exemplo o processo termina com a criação da aresta $\{v_3, v_{13}\}$.

A verificação de que esta construção é de facto a inversa da do código de Prüfer é deixada para os exercícios.

1.2 Grafos com pesos e árvores geradoras minimais

Em certas aplicações, é natural considerar grafos em que a cada aresta é atribuído um determinado valor real (usualmente positivo), um peso. Um problema relacionado com o anterior é o de determinar num grafo conexo com pesos, uma árvore geradora minimal, isto é, com peso total mínimo. Dois algoritmos que resolvem este problema, o de Boruvka-Kruskal e o de Jarník-Prim, baseiam-se no mesmo princípio "ganancioso" de escolher sucessivamente arestas com o menor peso possível.

No algoritmo de Boruvka-Kruskal, a partir de um vértice inicial arbitrário, escolhe-se em cada passo uma aresta de peso o menor possível que não forme um ciclo com outras arestas já incluídas; o algoritmo produz assim uma sucessão de **florestas** que termina com uma árvore geradora minimal.

No algoritmo de Jarník-Prim, também partindo de um vértice inicial arbitrário, escolhe-se em cada passo uma aresta com o menor peso possível, que seja incidente em um vértice já escolhido e noutro ainda não escolhido, ou seja, o algoritmo produz uma sucessão de **árvores**, que termina igualmente com uma árvore geradora minimal.

Para provar que o algoritmo de Boruvka-Kruskal produz uma árvore geradora minimal T, considere-se a lista $a_1, \cdots, a_{n-1}$ das suas arestas, ordenada pela ordem pela qual foram escolhidas, e uma árvore geradora minimal T'. Suponhamos que T' contém o bloco inicial $a_1, \cdots, a_{k-1}$ daquela lista de arestas mas não contém a aresta a_k.
Seja C o (único) caminho em T' que une os vértices incidentes a a_k; C pode conter algumas das arestas $a_1, \cdots, a_{k-1}$, mas tem que conter também arestas fora desse conjunto, caso contrário a_k faria um ciclo com arestas já escolhidas em passos anteriores do algoritmo; vamos ver que, de facto, todas as arestas de C que não estão no conjunto $\{a_1, \cdots, a_{k-1}\}$ têm peso igual ao de a_k:
se existisse alguma aresta a em C com peso menor que $p(a_k)$, ela teria sido escolhida em vez de a_k; note-se de facto que, como quer as arestas $a_1, \cdots, a_{k-1}$ quer a estão em T', a não pode criar um ciclo com aquelas.
Mas, por outro lado, se existisse alguma aresta a em C com peso maior do que $p(a_k)$, poderíamos substituir em T' a aresta a por a_k, obtendo uma árvore com peso menor.
Mas então esta substituição em T' da aresta a por a_k permite obter uma árvore geradora minimal contendo um bloco inicial maior da lista das arestas de T.
Repetindo este procedimento, acabamos por concluir que a própria árvore T é minimal.
Em alternativa, podíamos supor que T' era uma árvore geradora minimal contendo um bloco inicial $a_1, \cdots, a_{k-1}$ das arestas de T o maior possível e, através do raciocínio anterior, chegar a uma contradição.

Nota 1.13 *Pela sua relação com o tema desta secção, referimos aqui o conhecido "problema do Caixeiro-Viajante" que pode ser descrito como o problema de determinar, dado um grafo completo com pesos nas arestas, um ciclo Hamiltoniano (ou seja, um ciclo que passa por todos os vértices) de peso mínimo. Note-se que este problema contém, como caso particular, o de determinar se um grafo conexo qualquer tem um ciclo Hamiltoniano.*
Apesar da semelhança aparente com o problema de determinar uma árvore geradora minimal, este problema é muito mais complexo. Trata-se, de facto, de um dos vários problemas básicos da Teoria dos Grafos que são NP-completos.

Exercícios 2

1. Determinar a árvore com vértices $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ cujo código, pelo algoritmo de Prüfer, é o seu número de aluno, acrescentado de 3 algarismos à escolha.

2. Dados $1 < k < n$ fixos, calcular o número de árvores com vértices $\{v_1, \cdots, v_n\}$ tais que:

   a) $\{v_1, \cdots, v_k\}$ está contido no conjunto das folhas;

   b) $\{v_1, \cdots, v_k\}$ é o conjunto das folhas.

3. Seja a uma aresta de K_n. Mostrar que $K_n - a$ tem $(n-2)n^{n-3}$ árvores geradoras.

4. Mostrar que no código de Prüfer o índice do vértice v ocorre $d(v) - 1$ vezes.

5. Mostrar que se $[c_1, c_2, \cdots, c_{n-2}]$ é o código de Prüfer da árvore T, e v é a folha com menor índice, $[c_2, \cdots, c_{n-2}]$ é o código de Prüfer da árvore $T - v$.

6. Mostrar que os vértices e arestas são eliminados no processo de construção do código de Prüfer pela mesma ordem em que são acrescentados às listas V e E na operação inversa de construção da árvore a partir do código.

7. Dado um grafo conexo G e um vértice v_0, mostrar que existe uma árvore geradora T, tal que, para todo o vértice v, a distância entre v_0 e v em T é igual à distância entre v_0 e v em G.

8. Dado um grafo G conexo com mais do que 2 vértices, justificar que existe um par $x, y \in V_G$ tal que $G - \{x, y\}$ ainda é conexo.

9. Mostrar que o número de árvores com vértices $v_1, \cdots, v_n$ para as quais $d(v_i) = d_i$ é dado por

$$\frac{(n-2)!}{(d_1-1)! \cdots (d_n-1)!}$$

Sugestão: usar indução em n; notar que podemos assumir que $d_n = 1$. Usando o teorema multinomial, deduzir a fórmula de Cayley.

10. Uma segunda demonstração da Fórmula de Cayley usa o conceito de **ramificação**.

Definição 1.14 *: Uma ramificação é uma árvore com as arestas orientadas de tal modo que cada vértice é o vértice final de no máximo uma aresta.*

É fácil verificar que existe um único vértice (a raiz da ramificação) que não é vértice final de qualquer aresta.

Dado um conjunto de vértices $v_1, \cdots, v_n$, contamos o número de modos de criar uma ramificação, escolhendo uma sequência de $n-1$ arestas: em cada passo, diminuímos o número de componentes conexas numa unidade, e cada uma dessas componentes conexas tem que satisfazer a propriedade da ramificação; no passo k, temos então $n-k+1$ componentes e podemos escolher qualquer um dos n vértices para vértice inicial da aresta; mas o vértice final tem que ser a raiz de uma das outras $n-k$ componentes.
Mostrar que há

$$\prod_{k=1}^{n-1} n(n-k) = n^{n-1}(n-1)!$$

construções dessa forma, mas que cada ramificação é contada $(n-1)!$ vezes

Concluir portanto que existem n^{n-1} ramificações com vértices $v_1, \cdots, v_n$ e deduzir a fórmula de Cayley: a quantas ramificações corresponde uma árvore?

11. Seja $k > 0$ e T uma árvore qualquer com $k+1$ vértices, dos quais escolhemos um para raiz. Mostrar que se G é um grafo simples com grau mínimo k, para qualquer vértice v de G, existe um subgrafo de G isomorfo a T em que a raiz é v.
Sugestão: indução.

12. Dados dois conjuntos disjuntos

$$V = \{v_1, \cdots, v_m\}, \qquad U = \{u_1, \cdots, u_n\}$$

calcular o número de árvores com vértices $V \cup U$, em que cada aresta incide num vértice de V e noutro de U, contando as ramificações em $V \cup U$ com raiz em V.

13. Determinar uma árvore geradora minimal do grafo definido pela seguinte matriz de adjacência com pesos (a entrada A_{ij} é 0 se os vértices v_i ev_j não são adjacentes; caso contrário $A_{ij} = p$ em que p é o peso da aresta que liga os dois vértices):

$$\begin{pmatrix} 0 & 9 & 5 & 8 & 3 & 8 & 7 & 1 \\ 9 & 0 & 8 & 0 & 7 & 4 & 7 & 3 \\ 5 & 8 & 0 & 1 & 0 & 0 & 4 & 8 \\ 8 & 0 & 1 & 0 & 4 & 9 & 3 & 9 \\ 3 & 7 & 0 & 4 & 0 & 3 & 8 & 7 \\ 8 & 4 & 0 & 9 & 3 & 0 & 9 & 6 \\ 7 & 7 & 4 & 3 & 8 & 9 & 0 & 7 \\ 1 & 3 & 8 & 9 & 7 & 6 & 7 & 0 \end{pmatrix}$$

14. Mostrar que se os pesos das arestas de um grafo são todos distintos a árvore geradora com peso mínimo é única.

15. Adaptar o raciocínio usado na demonstração da validade do algoritmo de

Boruvka-Kruskal para demonstar que o algoritmo de Jarník-Prim produz também uma árvore geradora minimal.