Capítulo 8: Estruturas

Grande parte da informática lida com a representação e manipulação de informação. Para tal, foram concebidas várias estruturas para organizar blocos de dados de uma forma que facilite o seu armazenamento, pesquisa e recuperação. Há uma disciplina completa em boa parte dos currículos de computação chamada "estruturas de dados" que cobre como implementar estas estruturas em código. Neste livro, não vamos falar do código, mas sim das próprias estruturas abstratas. Este capítulo tem muitas imagens, que descrevem exemplos das várias estruturas de uma forma muito geral. Os conceitos aqui descritos mapeiam diretamente para código, quando for necessário pô-los em prática.

Há todo o tipo de estruturas de dados — vetores, listas ligadas, filas, pilhas, hashtables, e heaps, para citar alguns — mas quase todos eles resumem-se a um de dois tipos fundamentais de coisas: **grafos**, e **árvores**. Estas são as duas estruturas em que nos vamos concentrar neste capítulo. Um grafo é praticamente a estrutura mais geral que se pode imaginar: um monte de elementos de dados dispersos que estão de alguma forma relacionados uns com os outros. Quase todas as estruturas de dados imagináveis podem ser reformuladas como um tipo de grafo. As árvores são uma espécie de caso especial de grafos, mas também uma espécie de tópico em si próprias, assim como funções eram um tipo especial de relação, mas também um pouco diferentes. Uma árvore pode ser vista como um tipo de grafo que impõe condições extras especiais que fornecem alguns benefícios para percorrê-las.

8.1. Grafos

Em muitos aspectos, a forma mais elegante, simples e poderosa de representar conhecimento é por meio de um **grafo**. Um grafo é composto por um monte de pequenos pedaços de dados, cada um dos quais pode (ou não) ser ligado a cada um dos outros. Um exemplo está na Figura 8.1. Cada uma das ovais rotuladas é chamada um **vértice**, e as linhas entre elas são chamadas **arestas**. Cada vértice contém, ou não, uma aresta que o liga a cada um dos outros vértices. Poder-se-ia imaginar cada um dos vértices contendo vários atributos descritivos — talvez a oval de Matheus Nachtergaele tivesse informação sobre a sua data de nascimento, e a oval de Rio Grande do Sul informação sobre a sua área, número de municípios e população — mas estes não são tipicamente mostrados no diagrama. Tudo o que realmente importa, em termos de grafos, é que vértices ele contém, e quais são os que se conectam a que outros.

Os psicólogos cognitivos, que estudam os processos mentais internos da mente, já há muito tempo identificaram este tipo de estrutura como a principal forma em que as pessoas mentalmente armazenam e trabalham com informação. Afinal, se refletirmos um momento e nos perguntarmos "quais são as 'coisas' que estão na minha memória", uma resposta razoável é "bem, eu sei sobre um monte de coisas, e as propriedades dessas coisas, e as relações entre essas coisas". Se as "coisas" são vértices, e as "propriedades" são atributos desses vértices, e as "relações" são as arestas, temos precisamente a estrutura de um grafo. Os psicólogos deram a isto outro nome: uma **rede semântica**. Pensa-se que a miríade de conceitos que você registrou na memória

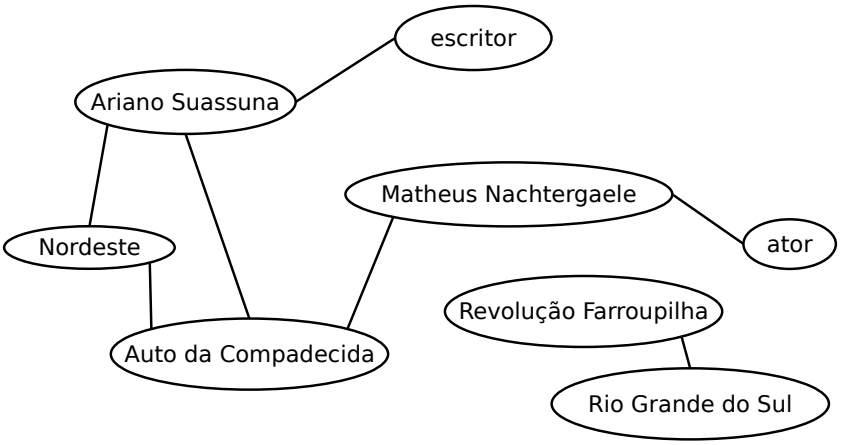

Figura 8.1: Um grafo (não direcionado).

— Ariano Suassuna, o desodorante, e seu calendário de aulas, e talvez milhões de outros — estão todos associados na sua mente numa vasta rede semântica que liga os conceitos relacionados entre si. Quando a sua mente recorda informações, ou deduz fatos, ou mesmo se desloca aleatoriamente em momentos de inatividade, está essencialmente a percorrer este grafo passando pelas várias arestas que existem entre os vértices.

Isso é profundo. Mas não é preciso ir tão fundo para ver o aparecimento de estruturas de grafos em toda a informática. O que é um mapa de um aplicativo de GPS, se não um grafo gigantesco onde os vértices são localizações transitáveis e as arestas são rotas entre eles? O que é uma rede social, se não um grafo gigante onde os vértices são pessoas e as arestas são amizades? O que é a World Wide Web, se não um grafo gigantesco onde os vértices são páginas e as arestas são hiperligações? O que é a Internet, se não um grafo enorme onde os vértices são computadores ou roteadores e as arestas são ligações de comunicação entre eles? Este esquema simples de vértices ligados é suficientemente poderoso para acomodar toda uma série de aplicações, e é por isso que vale a pena estudá-lo.

8.1.1 Termos de Grafos

O estudo dos grafos traz consigo toda uma série de novos termos que são importantes usar com precisão:

- **Vértice**. Cada grafo contém zero ou mais vértices (estes também são por vezes chamados de *nós*, *conceitos*, ou *objetos*)[1].
- **Aresta**. Cada grafo contém zero ou mais arestas (estas também são por vezes chamadas de *links*, *conexões*, *associações*, ou *relacionamentos*). Cada aresta liga exatamente dois vértices, a menos que a aresta ligue um vértice a si próprio, o que é possível, acredite ou não. Uma aresta que liga um vértice a si própria é chamada de *loop* ou *laço*.

1 A frase “zero ou mais” é comum em matemática discreta. Neste caso, ela indica que o grafo vazio, que não contém quaisquer vértices, continua a ser um grafo legítimo.

- **Caminho**. Um caminho é uma sequência de arestas consecutivas que o leva de um vértice para um outro. Na Figura 8.1, existe um caminho entre Nordeste e Matheus Nachtergaele (passando pelo Auto da Compadecida), embora não exista uma aresta direta entre os dois. Pelo contrário, não existe um caminho entre Ariano Suassuna e a Revolução Farroupilha. Não confunda os dois termos aresta e caminho: o primeiro é uma ligação direta entre dois nós, enquanto que o segundo pode ser uma travessia passo-a-passo inteira (uma aresta única, entretanto, conta como um caminho).
- **Direcionado/não direcionado** (ou **dirigido/não dirigido**). Em alguns grafos, os relacionamentos entre nós são inerentemente bidirecionais: se A está ligado a B, então B está ligado a A, e não faz sentido de outra forma. Pense numa rede social: a amizade vai sempre nos dois sentidos. Este tipo de grafo é chamado de grafo não direcionado, e tal como o exemplo de Ariano Suassuna na Figura 8.1, as arestas são mostradas como linhas retas. Noutras situações, uma aresta de A para B não implica necessariamente uma aresta também na direção inversa. Na World Wide Web, por exemplo, só porque a página A tem um link para a página B, não significa que o inverso seja verdade (normalmente não é). Neste tipo de grafo direcionado, desenhamos pontas de seta nas linhas para indicar para que sentido vai a ligação. Um exemplo é a Figura 8.2: os vértices representam pugilistas famosos, e as arestas direcionadas indicam que pugilista derrotou que outro(s). É possível que um par de vértices tenha arestas em ambas as direções — Muhammad Ali e Joe Frazier derrotaram cada um o outro (em momentos distintos, obviamente) — mas esta não é a norma, e certamente não é a regra, com um grafo direcionado.

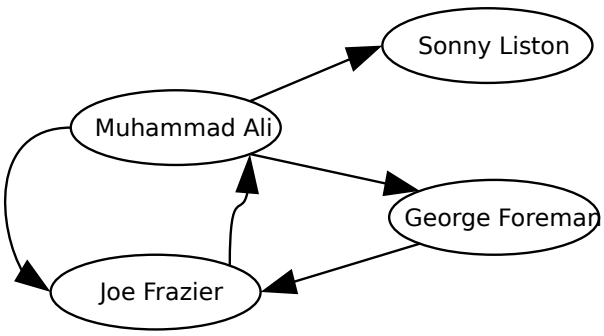

Figura 8.2: Um grafo direcionado.

- **Com pesos**. Alguns grafos, além de conterem apenas a presença (ou ausência) de uma aresta entre cada par de vértices, têm também um número em cada aresta, chamado de peso da aresta. Dependendo do grafo, este pode indicar a distância, ou custo, entre os vértices. Um exemplo está na Figura 8.3: à moda de aplicativos de navegação, este grafo contém as localizações, e a quilometragem entre elas. Um grafo pode também ser ao mesmo tempo direcionado e com pesos, a propósito. Se um par de vértices em tal grafo estiver ligado "nos dois sentidos", então cada uma das duas arestas terá o seu próprio peso.
- **Adjacente**. Se dois vértices tiverem uma aresta entre si, diz-se que são adjacentes.

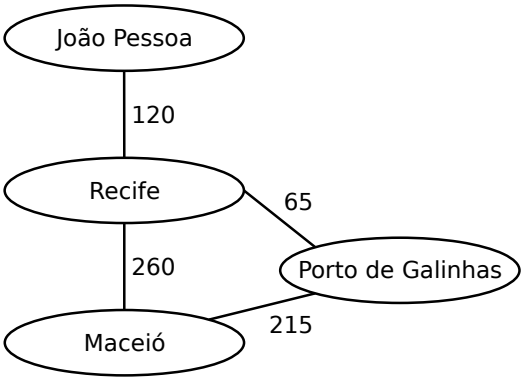

Figura 8.3: Um grafo (não direcionado) com pesos.

- **Conectividade**. A conectividade pode ter dois significados: aplica-se tanto a *pares de vértices* como a *grafos inteiros*. Dizemos que dois vértices estão **conectados** *se houver pelo menos um caminho entre eles*. Cada um destes vértices é, portanto, "*alcançável*" a partir do outro. Na Figura 8.1, Ariano Suassuna e ator estão conectados, mas o Auto da Compadecida e a Revolução Farroupilha não estão. Podemos também falar da conectividade de um grafo inteiro. Dizemos que um grafo *não direcionado* é **conexo**, se cada nó puder ser alcançado a partir de todos os outros. É fácil ver que a Figura 8.3 é um grafo conexo, enquanto que a Figura 8.1 não é (porque a Revolução Farroupilha e o Rio Grande do Sul estão isoladas dos outros nós). Para um grafo *direcionado*, dizemos que ele é conexo se a substituição de todas as suas arestas direcionadas por arestas não direcionadas produz um grafo (não-direcionado) conexo. Assim, o grafo da Figura 8.2 é conexo. Contudo, nem sempre é trivial determinar se um grafo é conexo: imagine um emaranhado de um milhão de vértices, com dez milhões de arestas, e ter que descobrir se cada vértice é ou não alcançável a partir de todos os outros (e se isso parecer irrealisticamente grande, considere uma rede social como o Facebook, que tem mais de um bilhão de nós).
- **Grau**. O grau de um vértice é simplesmente o número de arestas que se ligam a ele (loops contam duas vezes). Na Figura 8.3, Porto de Galinhas tem grau 2, e Recife, 3. No caso de um grafo dirigido, distinguimos por vezes entre o número de arestas de entrada que um vértice tem (chamado o seu grau de entrada) e o número de arestas de saída (o seu grau de saída). Muhammad Ali teve um grau de saída (3) superior ao grau de entrada (1), uma vez que ganhou a maior parte do tempo (cf. Figura 8.2).
- **Ciclo**. Um ciclo é um caminho que começa e termina no mesmo vértice.[2] Na Figura 8.3, Maceió—Porto de Galinhas—Recife—Maceió é um ciclo. Qualquer laço é um ciclo por si só. Para os grafos dirigidos, todo o ciclo deve compreender arestas na direção "para a frente": não há possibilidade de andar para trás.

2 Diremos também que um ciclo não pode repetir quaisquer arestas ou vértices pelo caminho, de modo que não possa andar para trás e para a frente repetidamente e sem sentido entre dois nós adjacentes. Alguns matemáticos chamam isto de um *ciclo simples* para o distinguir do ciclo mais geral, mas diremos apenas que nenhum ciclo pode repetir-se desta maneira.

Na Figura 8.2, Frazier—Ali—Foreman—Frazier é um ciclo, tal como o ciclo mais simples Ali—Frazier—Ali.

- **DAG** (grafo acíclico direcionado, do inglês *directed acyclic graph*). Um uso comum dos grafos é representar fluxos de dependências, por exemplo, os pré-requisitos que as diferentes disciplinas universitárias têm umas com as outras. Outro exemplo são os fluxos de tarefas na gestão de projetos: as tarefas necessárias para completar um projeto tornam-se vértices, e então as dependências que elas têm umas com as outras tornam-se arestas. O grafo da Figura 5.4 mostra os passos para fazer um bolo, e como estes passos dependem uns dos outros. Os ovos têm de ser quebrados antes dos ingredientes poderem ser misturados, e o forno tem de ser preaquecido antes de assar, mas a forma pode ser untada em qualquer ocasião, desde que seja feita antes de se despejar a massa nela.

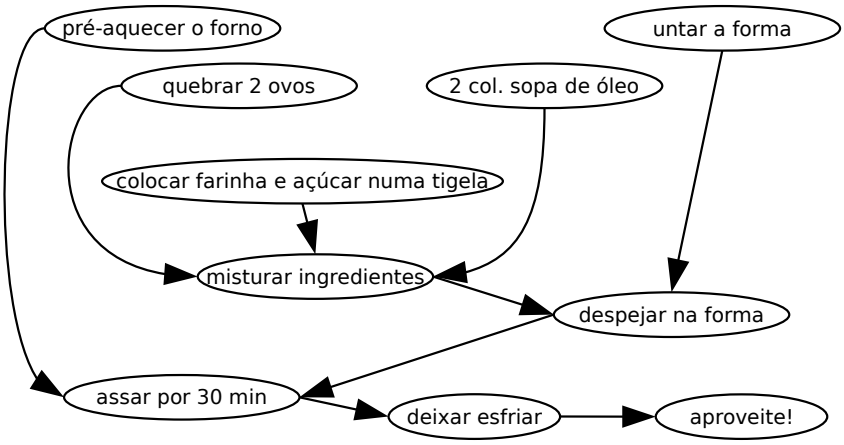

Figura 8.4: Um DAG.

Um grafo de dependências como este deve ser tanto **dirigido** como **acíclico**, ou não faria sentido. Dirigido, claro, significa que a tarefa X pode exigir que a tarefa Y seja concluída antes dela, sem que o inverso também seja verdadeiro. Se ambos dependessem um do outro, teríamos um loop infinito, e nenhum bolo poderia ser assado! Acíclico significa que *nenhum* tipo de ciclo pode existir no grafo, mesmo um que passe por múltiplos vértices. Tal ciclo resultaria novamente num loop infinito, tornando o projeto inútil. Imagine se houvesse uma seta de *assar por 30 minutos* de volta para *untar a forma* na Figura 8.4. Então, teríamos de untar a forma antes de despejar a massa nela, e teríamos que despejar a massa antes de assar, mas também teríamos de assar antes de untar a forma! Ficaríamos logo presos no trabalho: não haveria maneira de completar nenhuma dessas tarefas, uma vez que todas elas dependeriam indiretamente umas das outras. Um grafo que é tanto direcionado como acíclico (e portanto livre destes problemas) é por vezes chamado **DAG** para abreviar.

8.1.2 Posicionamento espacial

Uma coisa importante a compreender sobre os grafos é quais os aspectos de um diagrama que são relevantes. Especificamente, o *posicionamento espacial dos vértices não importa*. Na Figura 8.2 desenhamos Muhammad Ali do meio para a esquerda, e Sonny Liston no extremo superior direito. Mas esta foi uma escolha arbitrária, e irrelevante. Mais especificamente, isto não faz parte da informação que o diagrama pretende representar. Poderíamos ter posicionado os vértices de forma diferente, como na Figura 5.5, e obter o mesmo grafo. Em ambos os diagramas, há os mesmos vértices, e as mesmas arestas entre eles (verifique). Portanto, estes são matematicamente o mesmo grafo.

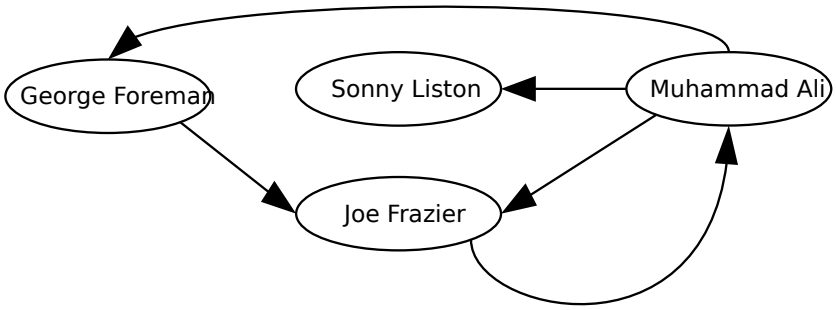

Figura 8.5: Uma visão diferente do mesmo grafo de boxeadores.

Isto pode não parecer surpreendente para o grafo dos boxeadores, mas para grafos como os de aplicações de GPS, em que nós na realidade representam localizações físicas, pode parecer chocante. Na Figura 8.3 poderíamos ter desenhado Recife ao norte de Maceió, e Porto de Galinhas num lugar qualquer do diagrama, e ainda ter o mesmo grafo, desde que todos os nós e ligações fossem os mesmos. Basta lembrar que o posicionamento espacial é concebido para conveniência humana, e não faz parte da informação matemática. É semelhante a como não há ordem para os elementos de um conjunto, embora quando especificamos uma extensão do conjunto, tenhamos de os listar em alguma ordem para evitar escrever todos os nomes dos elementos em cima uns dos outros. Num diagrama de um grafo, temos de desenhar cada vértice em algum lugar, mas onde os colocamos é simplesmente estético.

8.1.3 Relação com conjuntos

Parece que nos afastamos muito de conjuntos com todo este material sobre grafos. Mas na verdade, há algumas ligações importantes a serem feitas com esses conceitos iniciais. Recordemos o conjunto A de amigos do capítulo 2 que estendemos para conter {Helena, Ronaldo, Heitor, Nélson}. Consideremos agora a seguinte endorrelação em A:

(Helena, Ronaldo)
(Ronaldo, Helena)
(Ronaldo, Heitor)
(Ronaldo, Nélson)
(Heitor, Heitor)
(Nélson, Helena)

Esta relação, e tudo o que ela contém, é fielmente representada pelo grafo da Figura 8.6. Os elementos de A são os vértices, claro, e cada par ordenado da relação é refletido em uma aresta do grafo. Você consegue ver como exatamente a mesma informação é representada por ambas as formas?

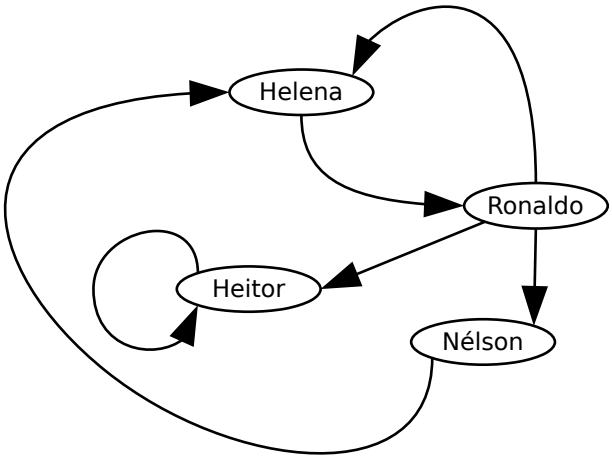

Figura 8.6: Um grafo representando uma endorrelação.

A Figura 8.6 é um grafo direcionado, é claro. E se fosse um grafo não direcionado? A resposta é que a relação correspondente seria *simétrica*. Um grafo não direcionado implica que se houver uma aresta entre dois vértices, ela "vai para os dois lados". Isto é de fato idêntico a dizer que uma relação é simétrica: se um (x, y) está na relação, então o correspondente (y, x) também deve estar. Um exemplo é a Figura 8.7, que retrata a seguinte relação simétrica:

(Helena, Ronaldo)
(Ronaldo, Helena)
(Ronaldo, Heitor)
(Heitor, Ronaldo)
(Helena, Helena)
(Nélson, Nélson)

Observe como os loops (arestas de um nó de volta para si próprio) nestes diagramas representam pares ordenados em que ambos os elementos são iguais.

Outra ligação entre grafos e conjuntos tem a ver com partições. A Figura 8.7 não era um grafo conexo: Nélson não podia ser alcançado a partir de nenhum dos outros nós. Agora considere: um grafo como este não é semelhante de alguma forma a uma partição de A — concretamente, esta?

{ Helena, Ronaldo, Heitor } e { Nélson }.

Simplesmente particionamos os elementos de A nos grupos que estão conectados. Se removermos a aresta entre Helena e Ronaldo nesse grafo, teremos:

{ Helena }, { Ronaldo, Heitor }, e { Nélson }.

Depois acrescente uma aresta entre Heitor e Nélson, e agora temos:

{ Helena } e { Ronaldo, Heitor, Nélson }.

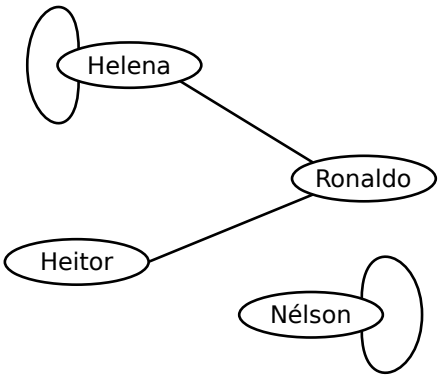

Figura 8.7: Um grafo representando uma endorrelação simétrica.

Em outras palavras, a "conectividade" de um grafo pode ser representada precisamente como uma partição do conjunto de vértices. Isto porquê cada subconjunto conectado, ou **componente conexo** do grafo, é um grupo, e cada vértice está num e apenas num grupo: por conseguinte, estes grupos isolados são mutuamente exclusivos e coletivamente exaustivos, ou seja, formam uma partição do conjunto de vértices. Legal.

8.1.4 Percursos em grafos

Se você tivesse uma longa lista — talvez de números de telefone, nomes, ou ordens de compra — e precisasse percorrê-la e fazer algo a cada elemento da lista — ligar para todos os números, procurar um determinado nome na lista, somar todas as ordens — seria bastante óbvio como fazê-lo. Basta começar pelo topo e trabalhar até o fim. Pode ser enfadonho, mas não é confuso.

Iterar pelos elementos desta forma chama-se **percorrer** a estrutura de dados. Você quer ter a certeza de encontrar cada elemento uma vez (e apenas uma vez) para poder fazer o que for preciso com ele. É evidente como se percorre uma lista. Mas como percorrer um grafo? Não há um "primeiro" ou "último" nó óbvio, e cada um deles está potencialmente ligado a muitos outros. E como vimos, os vértices podem nem sequer estar totalmente conectados, pelo que um caminho que percorre todos os nós pode nem sequer existir.

Há duas formas diferentes de percorrer um grafo: em largura, e em profundidade. Elas fornecem formas diferentes de explorar os nós, e como efeito secundário, cada uma é capaz de descobrir se o grafo é ou não conexo. Vamos examinar cada uma delas a seguir.

Travessia em Largura

Com a travessia em largura, começamos num vértice inicial (não importa qual) e exploramos o gráfico cautelosamente e delicadamente. Investigamos igualmente a fundo em todas as direções, certificando-nos de que olhamos um pouco para cada caminho possível antes de explorarmos cada um desses caminhos um pouco mais longe.

Para tal, usamos uma estrutura de dados muito simples chamada **fila**. Uma fila é simplesmente uma lista de nós que estão à espera na fila. Quando inserimos um nó na fi-

nal da fila, dizemos que “**enfileiramos o nó**”, e quando retiramos um nó da frente da fila, dizemos que “**desenfileiramos o nó**”. Os nós do meio esperam pacientemente chagar sua vez de serem tratados, aproximando-se da frente da fila cada vez que o nó da frente é desenfileirado.

Um exemplo desta estrutura de dados em ação é mostrado na Figura 5.8. Note cuidadosamente que inserimos sempre os nós numa extremidade (à direita) e os retiramos da outra extremidade (à esquerda). Isto significa que o primeiro item a ser enfileirado (neste caso, o triângulo) será o primeiro a ser desenfileirado. “As chamadas serão respondidas na ordem em que foram recebidas”. Este fato deu origem a outro nome para uma fila: uma “**FIFO**”, que significa “primeiro a entrar e primeiro a sair, do inglês *first-in-first-out*”..

Comece com uma fila vazia:	|
Enfileire um triângulo, então temos:	|△
Enfileire uma estrela, então temos:	|△✶
Enfileire um coração, então temos:	|△✶♡
Desenfileire (o triângulo), então temos:	|✶♡
Enfileire um trevo, então temos:	|✶♡♣
Desenfileire (a estrela), então temos:	|♡♣
Desenfileire (o coração), então temos:	|♣
Desenfileire (o trevo). Estamos vazios novamente:	|

Figura 8.8: Uma fila em ação. A barra vertical marca a "frente da fila", e os elementos estão à espera de serem desenfileirados, em ordem, da esquerda para a direita.

Agora, eis como utilizamos uma fila para percorrer um grafo em largura. Vamos começar por um nó em particular, e colocar todos os seus nós adjacentes numa fila. Isto faz com que todos eles fiquem “à espera na fila” em segurança até os explorarmos. Depois, repetidamente pegamos o primeiro nó da fila, fazemos o que precisamos fazer com ele, e depois colocamos todos os seus nós adjacentes na fila. Continuamos a fazer isto até que a fila esteja vazia.

Agora pode ter-lhe ocorrido que podemos ter problemas se encontrarmos o mesmo nó várias vezes enquanto estamos a percorrer. Isto pode acontecer se o grafo tiver um ciclo: haverá mais de um caminho para chegar a alguns nós, e podemos ficar presos num loop infinito se não tivermos cuidado. Por esta razão, introduzimos o conceito de **marcação de nós**. Isto é como deixar um rastro de migalhas de pão: se alguma vez estivermos prestes a explorar um nó, mas descobrirmos que ele está marcado, então sabemos que já lá estivemos, e é inútil pesquisá-lo novamente.

Portanto, há duas coisas que vamos fazer aos nós enquanto buscamos:

- **Marcar** um nó significa recordar que já o encontramos no processo da nossa pesquisa.

- **Visitar** um nó significa realmente fazer o que quer que seja que precisamos fazer com o nó (ligar para o número de telefone, examinar o seu nome para um casamento de padrão, adicionar o número ao nosso total, o que quer que seja).

A travessia em largura (**BFT** — do inglês *breadth-first traversal*) é um algoritmo, o que significa simplesmente um procedimento passo-a-passo, fiável, que garante a produção de um resultado. Neste caso, é garantido que se visite todos os nós do grafo que são alcançáveis a partir do nó inicial, e que não se fique preso em quaisquer loops infinitos no processo. Aqui está ele:

Travessia em Largura (BFT)

1. Escolher um nó de partida.
2. Marque-o e enfileire-o numa fila até então vazia.
3. Enquanto a fila não estiver vazia, faça estes passos:
   a. Desenfileire o nó do início da fila.
   b. Visite-o.
   c. Marque e enfileire todos os *nós não marcados adjacentes a ele* (em qualquer ordem)

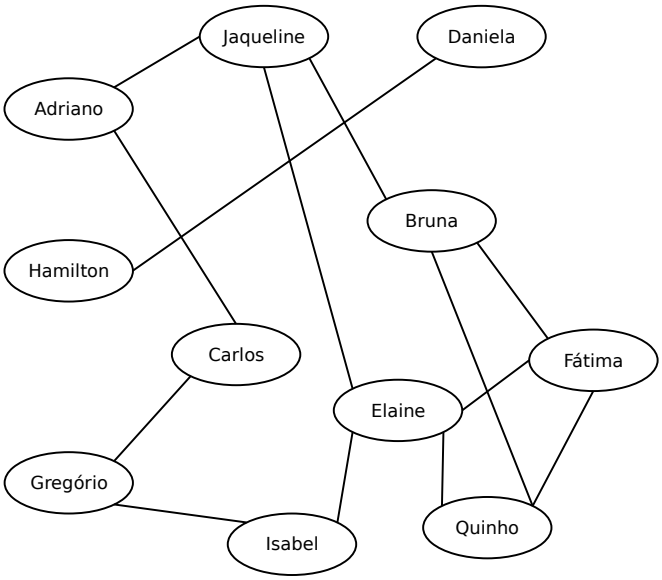

Figura 8.9: Grupo de pessoas numa rede social e suas relações de amizade.

Vamos executar este algoritmo e vê-lo em ação num conjunto de utilizadores de uma rede social. A Figura 8.9 retrata onze utilizadores, e as amizades entre eles. Agora acompanhemos a execução do algoritmo BFT neste grafo na Figura 8.10. Primeiro, escolhemos Gregório como o nó de partida (não por nenhuma razão em particular, apenas porque temos de começar em algum lugar). Marcamo-lo (cinza claro no diagrama) e colocamo-lo na fila (o conteúdo da fila é listado na parte inferior de cada quadro, com a frente da fila à esquerda). Depois, começamos o nosso laço. Quando ti-

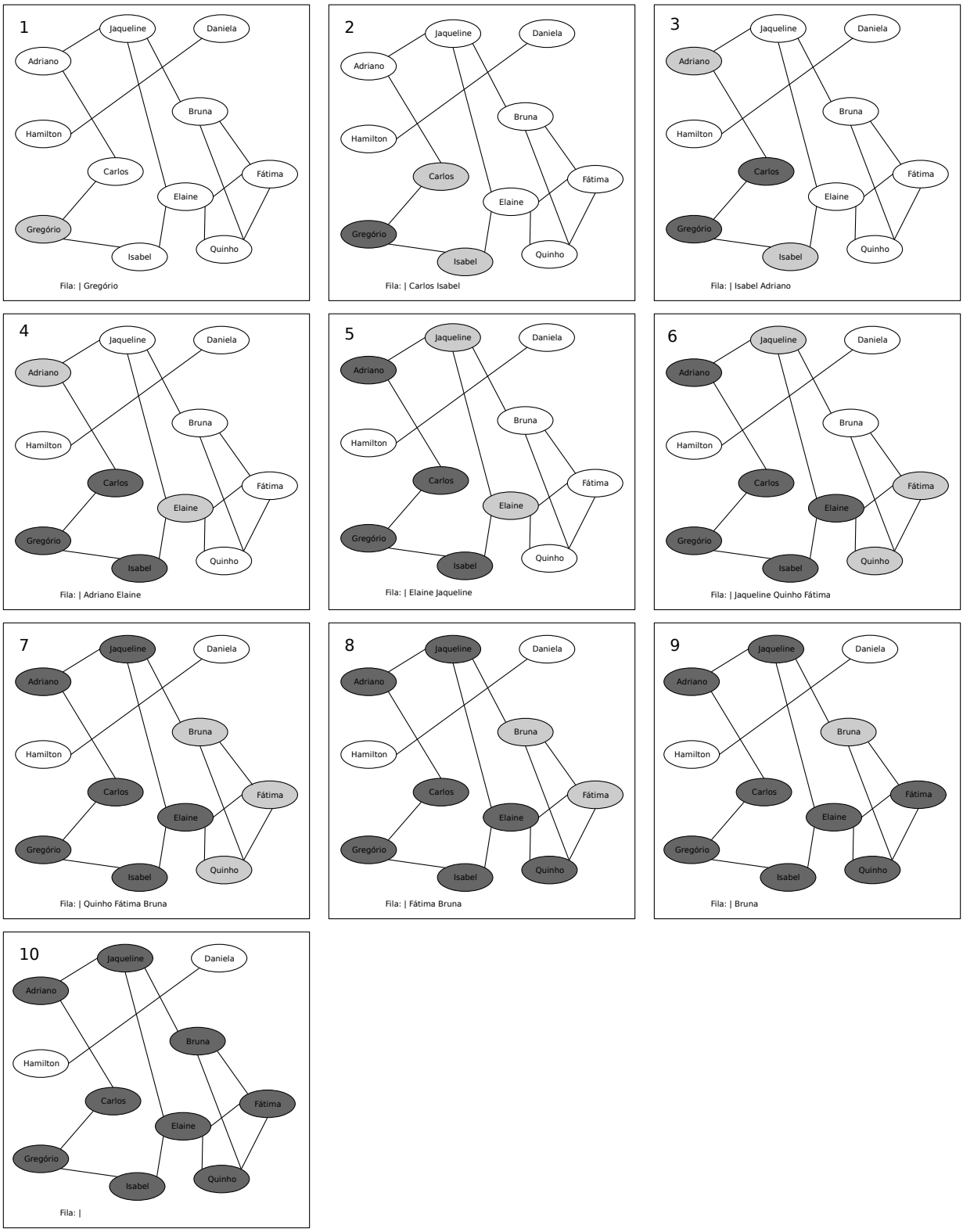

Figura 8.10: As etapas da travessia em largura. Os nós marcados são cinza claro, e os nós visitados são cinza escuro. A ordem de visita é: G, C, I, A, E, J, Q, F, B.

ramos o Gregório da fila, visitamo-lo (o que significa que "fazemos o que for preciso fazer ao Gregório") e depois marcamos e enfileiramos os seus nós adjacentes Carlos e

Isabel. Não importa em que ordem os colocamos na fila, tal como não importava com que nó começamos. No quadro 3, Carlos foi desenfileirado, visitado, e os seus nós adjacentes colocados na fila. Apenas um nó é enfileirado aqui — Adriano — porque obviamente Gregório já foi marcado (e mesmo visitado, não menos) e esta marcação permite-nos ser inteligentes e não o reenfileiramos.

É neste ponto que a característica "em largura" se torna aparente. Acabamos de terminar com Carlos, mas em vez de explorarmos Adriano a seguir, retomamos com Isabel. Isto porque ela tem estado pacientemente à espera na fila, e chegou a sua vez. Assim, colocamos Adriano de lado (na fila, claro) e visitamos Isabel, enfileirando a sua vizinha Elaine no processo. Depois, voltamos a Adriano. O processo segue, seguindo a ordem em que os elementos tornam-se presentes na fila. A visita a Jaqueline leva-nos a enfileirar Bruna, e depois, quando tiramos Quinho da fila, não voltamos a enfileirar Bruna porque ela foi marcada e por isso sabemos que ela já tinha sido tratada.

Continuamos a visitar os nós da fila até a fila ficar vazia. Como podem ver, Hamilton e Daniela não serão visitados neste processo: isto porque aparentemente ninguém que eles conhecem conhece ninguém da turma de Gregório, e por isso não há maneira de os alcançar a partir de Gregório. Foi isto que quis dizer anteriormente ao falar que, como efeito secundário, o algoritmo BFT diz-nos se o gráfico é ou não conexo. Tudo o que temos de fazer é começar em algum lugar, rodar o BFT, e depois ver se algum nó não foi marcado e visitado. Se houver algum, e quisermos visitar todos os nós, podemos continuar com outro ponto de partida, e depois repetir o processo.

Travessia em Profundidade

Com a **travessia em profundidade**, ou **DFT** (do inglês *depth-first traversal*), exploramos o gráfico de forma arrojada e imprudente. Escolhemos a primeira direção que vemos, e mergulhamos até as suas profundezas, antes de recuarmos relutantemente e tentarmos os outros caminhos a partir do início.

O algoritmo é quase idêntico ao BFT, exceto que, em vez de uma fila, utilizamos uma **pilha**. Uma pilha é semelhante a uma fila, exceto que em vez de colocar elementos numa extremidade e retirá-los da outra, adiciona-se e retira-se da mesma extremidade. Esta "extremidade" é chamada de **topo da pilha**. Quando adicionamos um elemento a este extremo, dizemos que o empurramos na pilha, e quando retiramos o elemento superior, dizemos que o tiramos do topo da pilha.

Pode-se pensar numa pilha como… bem, uma pilha, seja de livros ou de bandejas de cafeteria ou qualquer outra coisa. Não se pode tirar nada do meio de uma pilha, mas pode-se retirar itens e colocar mais itens, sempre do topo. A Figura 5.10 mostra um exemplo. O primeiro item que foi empurrado é sempre o último a ser retirado, e o mais recente está sempre pronto a ser tirado, e por isso uma pilha é por vezes também chamada "**LIFO**" (do inglês, *last-in-first-out*).

O próprio algoritmo de travessia em profundidade parece um déjà vu. Tudo o que se faz é substituir "fila" por "pilha":

Comece com uma pilha vazia:	__
Empurre um triângulo, então temos:	△ __
Empurre uma estrela, então temos:	✶ △ __
Empurre um coração, então temos:	♡ ✶ △ __
Retire (o coração), então temos:	✶ △ __
Empurre um trevo, então temos:	♣ ✶ △ __
Retire (o trevo), então temos:	✶ △ __
Retire (a estrela), então temos:	△ __
Retire (o triângulo). Estamos vazios novamente:	__

Figura 8.11: Uma pilha em ação. A barra horizontal marca o "fundo da pilha", e os elementos são colocados e tirados do topo.

Travessia em Profundidade (DFT)

1. Escolher um nó de partida.
2. Marque-o e empurre-o numa pilha até então vazia.
3. Enquanto a pilha não estiver vazia, faça estes passos:
    d. Retire o nó do topo da pilha.
    e. Visite-o.
    f. Marque e enfileire todos os *nós não marcados adjacentes a ele* (em qualquer ordem)

O algoritmo em ação é mostrado na Figura 8.12. A pilha fez realmente a diferença! Em vez de explorar alternadamente os caminhos de Carlos e de Isabel, ele mergulha de cabeça pelo caminho de Carlos até onde pode ir, até atingir a porta dos fundos de Isabel. Só depois volta atrás e visita Isabel. Isto porque a pilha sempre tira o que acabou de empurrar, enquanto que o que foi empurrado primeiro tem de esperar até que todo o resto seja feito antes de ter a sua oportunidade. Esse primeiro par de empurrões foi crítico: se tivéssemos empurrado Carlos antes de Isabel logo no início, então

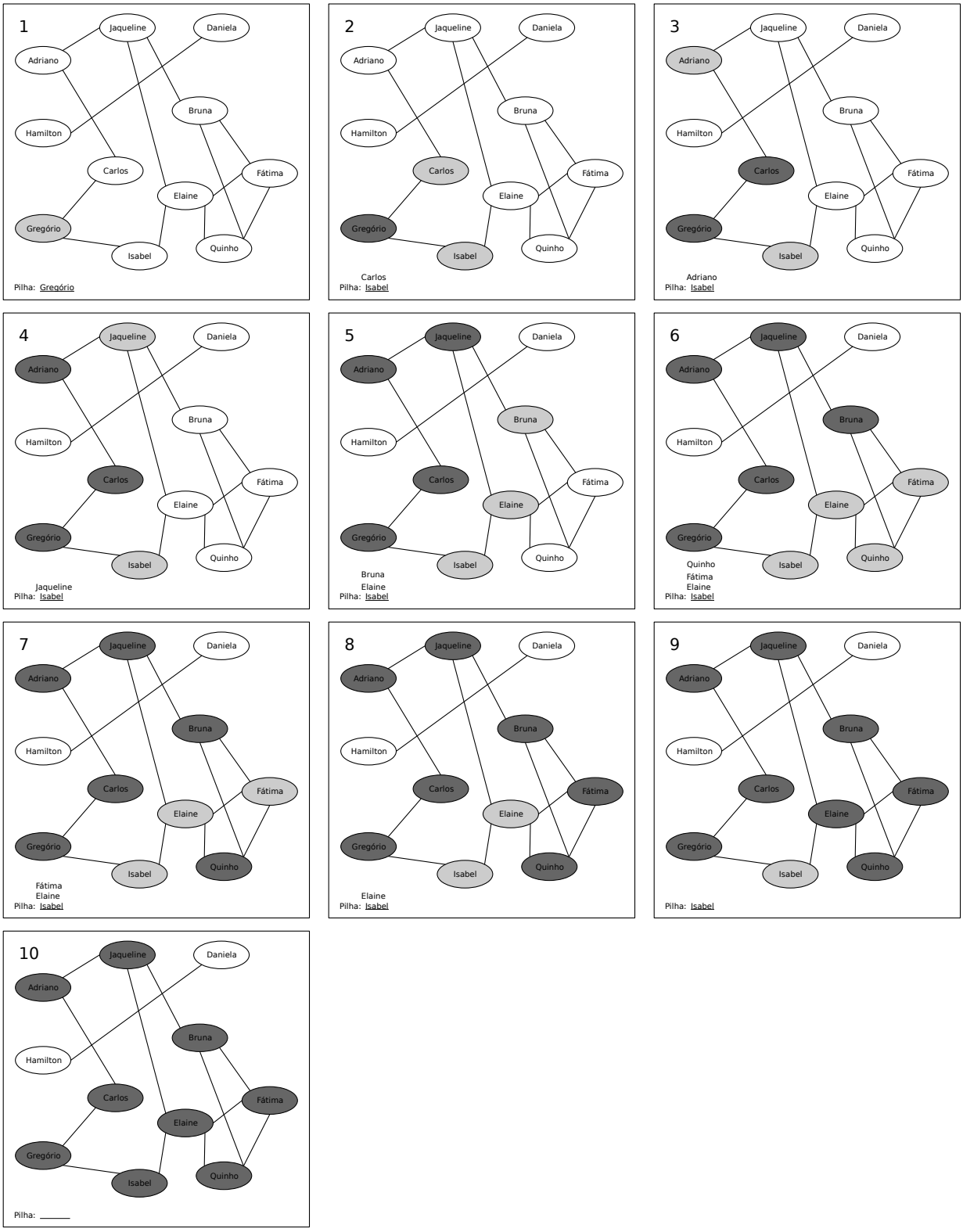

Figura 8.12: As etapas da travessia em profundidade. Os nós marcados são cinza claro, e os nós visitados são cinza escuro. A ordem de visita é: G, C, A, J, B, Q, F, E, I.

teríamos explorado todo o mundo de Isabel antes de chegarmos à porta dos fundos

de Carlos, em vez do contrário. Tal como está, Isabel foi colocada no fundo, e assim ficou no fundo, o que é inevitável com uma pilha.

O DFT identifica gráficos desconexos da mesma forma que o BFT, e evita igualmente ficar preso em loops infinitos quando encontra ciclos. A diferença fundamental é a ordem em que visita os nós.

8.2. Árvores

FIXME ainda vai ser acrescentado. Baixe novamente quando estiver pronto.

8.3. Créditos

Todas as seções, foram adaptadas (traduzidas e modificadas) de [1], que está disponível sob a licença Creative Commons Attribution-ShareAlike 4.0 International.

8.4. Referências

1. Davies, Stephen. *A Cool Brisk Walk Through Discrete Mathematics*. Disponível em http://www.allthemath.org/vol-i/

8.5. Licença