

grokking

# algorithms

Upcoming editions

Aditya Bhattacharya  
Harvard University

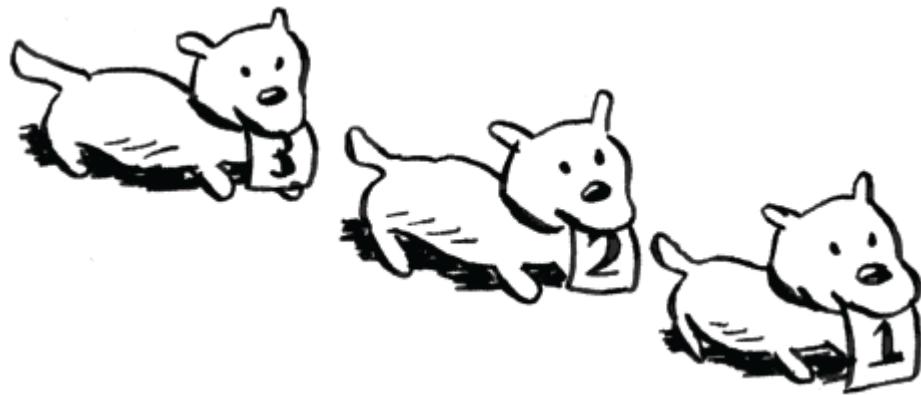


***inside front cover***

---

***cubierta frontal interior***

---



grokking

# algorithms

SECOND EDITION

Aditya Y. Bhargava

Foreword by Daniel Zingaro



## **Praise for the First Edition**

---

### **Elogios a la primera edición**

---

*"This book does the impossible: it makes math fun and easy!"*

*"Este libro hace lo imposible: ihace que las matemáticas sean divertidas y fáciles!"*

—Sander Rossel, COAS Software Systems

—Sander Rossel, Sistemas de software COAS

*"Do you want to treat yourself to learning algorithms in the same way that you would read your favorite novel? If so, this is the book you need!"*

*"¿Quieres darte el lujo de aprender algoritmos de la misma manera que leerías tu novela favorita? Si es así, ieste es el libro que necesitas!"*

—Sankar Ramanathan, IBM Analytics

—Sankar Ramanathan, IBM Analytics

*"In today's world, there is no aspect of our lives that isn't optimized by some algorithm. Let this be the first book you pick up if you want a well-explained introduction to the topic."*

*"En el mundo actual, no hay ningún aspecto de nuestras vidas que no esté optimizado mediante algún algoritmo. Deje que este sea el primer libro que elija si desea una introducción bien explicada al tema".*

—Amit Lamba, Tech Overture, LLC

—Amit Lamba, Tech Overture, LLC

*"Algorithms are not boring! This book was fun and insightful for both my students and me."*

*"Los algoritmos no son aburridos! Este libro fue divertido y revelador tanto para mis alumnos como para mí".*

—Christopher Haupt, Mobirobo, Inc

—Christopher Haupt, Mobirobo, Inc.



# *Grokking Algorithms*

# *Algoritmos de asimilación*

SECOND EDITION

SEGUNDA EDICION

**Aditya Y. Bhargava**  
**Aditya Y. Bhargava**

FOREWORD BY DANIEL ZINGARO  
PRÓLOGO DE DANIEL ZINGARO

To comment go to [liveBook](#)

Para comentar ve al liveBook



**Manning**  
**Manning**  
**Shelter Island**  
**Isla Refugio**

For more information on this and other Manning titles go to  
Para obtener más información sobre este y otros títulos de Manning, visite  
[www.manning.com](http://www.manning.com)  
[www.manning.com](http://www.manning.com)

## **Copyright**

### **Derechos de autor**

For online information and ordering of these and other Manning books, please visit [www.manning.com](http://www.manning.com). The publisher offers discounts on these books when ordered in quantity.

Para obtener información en línea y realizar pedidos de estos y otros libros de Manning, visite [www.manning.com](http://www.manning.com). La editorial ofrece descuentos en estos libros cuando se solicitan en cantidad.

For more information, please contact

Para obtener más información, póngase en contacto

Special Sales Department  
Departamento de Ventas Especiales  
Manning Publications Co.  
Publicaciones Manning Co.  
20 Baldwin Road  
20 Baldwin Road  
PO Box 761  
Apartado postal 761  
Shelter Island, NY 11964  
Isla Shelter, Nueva York 11964  
Email: [orders@manning.com](mailto:orders@manning.com)  
Correo electrónico: [pedidos@manning.com](mailto:pedidos@manning.com)

**©2024 by Manning Publications Co. All rights reserved.**

**©2024 por Manning Publications Co. Todos los derechos reservados.**

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Ninguna parte de esta publicación puede reproducirse, almacenarse en un sistema de recuperación ni transmitirse, de ninguna forma o por medios electrónicos, mecánicos, fotocopias o de otro tipo, sin el permiso previo por escrito del editor.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Muchas de las designaciones utilizadas por fabricantes y vendedores para distinguir sus productos se consideran marcas comerciales. Cuando esas designaciones aparecen en el libro y Manning Publications tenía conocimiento de un reclamo de marca registrada, las designaciones se imprimieron en mayúsculas iniciales o todas en mayúsculas.

⊖ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

⊖ Reconociendo la importancia de preservar lo que se ha escrito, la política de Manning es imprimir los libros que publicamos en papel libre de ácido, y hacemos nuestros mejores esfuerzos para lograr ese fin. Reconociendo también nuestra responsabilidad de conservar los recursos de nuestro planeta, los libros de Manning se imprimen en papel que al menos un 15 por ciento es reciclado y procesado sin el uso de cloro elemental.



Manning Publications Co.  
Publicaciones Manning Co.

20 Baldwin Road Technical  
20 Baldwin Road Técnico

PO Box 761  
Apartado postal 761

Shelter Island, NY 11964  
Isla Shelter, Nueva York 11964

Development editor: Ian Hough  
Editor de desarrollo: Ian Hough

Technical editor: David Eisenstat  
Editor técnico: David Eisenstat

Review editor: Aleksandar Dragosavljević  
Editor de reseñas: Aleksandar Dragosavljević

Production editor: Andy Marinkovich  
Redactor de producción: Andy Marinkovich

Copy editor: Alisa Larson  
Editor de copia: Alicia Larson

Proofreader: Jason Everett  
Corrector de pruebas: Jason Everett

Technical proofreader: Tony Holdroyd  
Corrector técnico: Tony Holdroyd

Typesetter: Dennis Dalinnik  
Tipógrafo: Dennis Dalinnik

Cover designer: Leslie Haimes  
Diseñador de la portada: leslie haimes

ISBN: 9781633438538

ISBN: 9781633438538

## **Dedication**

---

## **Dedicación**

---

*For my parents, Sangeeta and Yogesh*

*Para mis padres, Sangeeta y Yogesh.*

# **contents**

---

## **contenido**

---

*Front matter*

**Asunto frontal**

foreword

prefacio

preface

prefacio

acknowledgments

expresiones de gratitud

about this book

sobre este libro

about the author

Sobre el Autor

**1 introduction to algorithms**

**1 introducción a los algoritmos**

[Binary search](#)

Búsqueda binaria

[Big O notation](#)

Notación O grande

## **2 *selection sort***

### **2 *clasificación de selección***

[How memory works](#)

como funciona la memoria

[Arrays and linked lists](#)

Matrices y listas enlazadas

[Which is used more, arrays or linked lists?](#)

¿Qué se utiliza más, los arrays o las listas enlazadas?

[Selection sort](#)

Orden de selección

[Example code listing](#)

Listado de códigos de ejemplo

## **3 *recursion***

### **3 *recursividad***

Recursion  
recursividad  
Base case and recursive case  
Caso base y caso recursivo  
The stack  
La pila

## 4 *quicksort*

### 4 *clasificación rápida*

Divide and conquer  
Divide y conquistarás  
Quicksort  
Ordenación rápida  
Big O notation revisited  
Notación Big O revisada

## 5 *hash tables*

### 5 *tablas hash*

[Hash functions](#)

[Funciones hash](#)

[Use cases](#)

[Casos de uso](#)

[Collisions](#)

[Colisiones](#)

[Performance](#)

[Actuación](#)

## **6 *breadth-first search***

### **6 *búsqueda en amplitud***

[Introduction to graphs](#)

[Introducción a los gráficos.](#)

[What is a graph?](#)

[¿Qué es una gráfica?](#)

[Breadth-first search](#)

[Búsqueda en amplitud](#)

[Implementing the graph](#)

[Implementando el gráfico](#)

[Implementing the algorithm](#)

[Implementando el algoritmo](#)

## **7 *trees***

### **7 *árboles***

[Your first tree](#)

[Tu primer árbol](#)

[A space odyssey: Depth-first search](#)

[Una odisea espacial: búsqueda en profundidad](#)

[Binary trees](#)

[árboles binarios](#)

[Huffman coding](#)

[Codificación Huffman](#)

## **8 *balanced trees***

### **8 árboles equilibrados**

[A balancing act](#)

[Un acto de equilibrio](#)

[Shorter ...](#)

[Corta ...](#)

# ***front matter***

---

## ***asunto frontal***

---

### ***foreword***

### ***prefacio***

More people than ever need to learn how to program. Sure, some people literally program for their jobs (software engineers or web developers, for example). But many other jobs, not historically requiring programming, have a programming component now or will in the future. Programming also helps people understand the technological world in which they live.

Más personas que nunca necesitan aprender a programar. Claro, algunas personas literalmente programan para sus trabajos (ingenieros de software o desarrolladores web, por ejemplo). Pero muchos otros trabajos, que históricamente no requerían programación, tienen un componente de programación ahora o lo tendrán en el futuro. La programación también ayuda a las personas a comprender el mundo tecnológico en el que viven.

Unfortunately, the benefits of programming are not equally distributed. In North American computer science (CS) programs, for example, we have a very low participation of women and some ethnic/racial groups. It's critical that we

be able to expand programming and CS to a more diverse group. The solution will involve making progress on a number of fronts, including overcoming bias, training more teachers, and offering more diversified learning experiences. We need to help more people “get in.”

Lamentablemente, los beneficios de la programación no se distribuyen equitativamente. En los programas de informática (CS) de América del Norte, por ejemplo, tenemos una participación muy baja de mujeres y de algunos grupos étnicos/raciales. Es fundamental que podamos ampliar la programación y la informática a un grupo más diverso. La solución implicará avanzar en varios frentes, incluida la superación de prejuicios, la formación de más docentes y la oferta de experiencias de aprendizaje más diversificadas. Necesitamos ayudar a más personas a “entrar”.

I'm excited about Bhargava's book because it offers a new way to get into algorithms, which is a key component of effective programming. Some people will tell you that there's only one way to learn algorithms: find a dense mathematical book about algorithms, read it, and, like, understand everything. But that privileges the types of people who can learn that way, who have time to learn that way, and who need to learn that way in the first place. It also assumes that we know *why* someone wants to learn algorithms, which, let's face it, is not a fair assumption to make.

Estoy entusiasmado con el libro de Bhargava porque ofrece una nueva forma de profundizar en los algoritmos, que son un componente clave de una programación eficaz. Algunas personas te dirán que sólo hay una manera de aprender algoritmos: encontrar un libro matemático denso sobre algoritmos, leerlo y entenderlo todo. Pero eso privilegia al tipo de personas que pueden aprender de esa manera, que tienen tiempo para aprender de esa manera y que necesitan aprender de esa manera en primer lugar. También supone que sabemos por qué alguien quiere aprender algoritmos, lo cual, seamos realistas, no es una suposición justa.

To be clear, some of my favorite CS books are exactly those kinds of mathematically oriented algorithms books. Those books work for me. They work for a lot of CS professors. But maybe that's the problem: it's too easy to assume that the way we learn is the same way that others learn. What we need are all kinds of learning resources about all kinds of CS topics, each designed for a particular audience.

Para ser claros, algunos de mis libros de informática favoritos son exactamente ese tipo de libros de algoritmos orientados matemáticamente. Esos libros funcionan para mí. Trabajan para muchos profesores de informática. Pero tal vez ese sea el problema: es demasiado fácil asumir que la forma en que aprendemos es la misma que aprenden los demás. Lo que necesitamos son todo tipo de recursos de aprendizaje sobre todo tipo de temas de informática, cada uno diseñado para una audiencia particular.

Bhargava's book is intentionally designed for people who want a nonmathematical introduction to algorithms. What impresses me most here is not what Bhargava chose to include but what he chose *not* to include. You can't include everything in a book like this—that would be overwhelming and is not the point.

El libro de Bhargava está diseñado intencionalmente para personas que desean una introducción no matemática a los algoritmos. Lo que más me impresiona aquí no es lo que Bhargava decidió incluir sino lo que decidió no incluir. No se puede incluir todo en un libro como este; eso sería abrumador y no es el punto.

Bhargava's teaching expertise enables him to wring a lot of teaching out of not a lot of pages. In reading the "Dynamic Programming" chapter, for example, I was struck by the care with which Bhargava answers a lot of anticipated reader questions that other algorithm books would not answer.

La experiencia docente de Bhargava le permite extraer mucha enseñanza de pocas páginas. Al leer el capítulo "Programación dinámica", por ejemplo, me sorprendió el cuidado con el que Bhargava responde a muchas preguntas anticipadas de los lectores que otros libros sobre algoritmos no responderían.

I hope that this book helps you learn, whether you're trying algorithms for the first time or you've struggled to find the right resource until now. Happy Grokking!

Espero que este libro le ayude a aprender, ya sea que esté probando algoritmos por primera vez o que haya tenido dificultades para encontrar el recurso adecuado hasta ahora. ¡Feliz Grokking!

—Daniel Zingaro, University of Toronto

—Daniel Zingaro, Universidad de Toronto

## ***preface***

## ***prefacio***

I first got into programming as a hobby. *Visual Basic 6 for Dummies* taught me the basics, and I kept reading books to learn more. But the subject of algorithms was impenetrable for me. I remember savoring the table of contents of my first algorithms book, thinking “I’m finally going to understand these topics!” But it was dense stuff, and I gave up after a few weeks. It wasn’t until I had my first good algorithms professor that I realized how simple and elegant these ideas were.

Primero me metí en la programación como hobby. Visual Basic 6 para Dummies me enseñó los conceptos básicos y seguí leyendo libros para aprender más. Pero el tema de los algoritmos era impenetrable para mí. Recuerdo saborear el índice de mi primer libro de algoritmos y pensar: "¡Finalmente voy a entender estos temas!". Pero era algo denso y lo dejé después de unas semanas. No fue hasta

que tuve mi primer buen profesor de algoritmos que me dice cuenta de lo simples y elegantes que eran estas ideas.

I wrote my first illustrated blog post back in 2012. I'm a visual learner, and I really liked the illustrated style. Since then, I've written a few illustrated posts on functional programming, Git, machine learning, and concurrency. By the way, I was a mediocre writer when I started out.

Explaining technical concepts is hard. Coming up with good examples takes time, and explaining a difficult concept takes time. So it's easiest to gloss over the hard stuff. I thought I was doing a pretty good job until after one of my posts got popular, a coworker came up to me and said, "I read your post, and I still don't understand this." I still had a lot to learn about writing.

Escribí mi primera publicación de blog ilustrada en 2012. Aprendo visualmente y realmente me gustó el estilo ilustrado. Desde entonces, he escrito algunas publicaciones ilustradas sobre programación funcional, Git, aprendizaje automático y concurrencia. Por cierto, yo era un escritor mediocre cuando comencé. Explicar conceptos técnicos es difícil. Dar buenos ejemplos lleva tiempo y explicar un concepto difícil lleva tiempo. Por eso es más fácil pasar por alto las cosas difíciles. Pensé que estaba haciendo un buen trabajo hasta que una de mis publicaciones se hizo popular, un compañero de trabajo se me acercó y me dijo: "Leí tu publicación y todavía no entiendo esto". Todavía tenía mucho que aprender sobre la escritura.

Somewhere in the middle of writing these blog posts, Manning reached out to me and asked if I wanted to write an illustrated book. Well, it turns out that Manning editors know a lot about explaining technical concepts, and they taught me how to teach. I wrote this book to scratch a particular itch: I wanted to write a book that explained hard technical topics well, and I wanted an easy-to-read algorithms book.

En algún momento mientras escribía estas publicaciones de blog, Manning se acercó a mí y me preguntó si quería escribir un libro ilustrado. Bueno, resulta que los editores de Manning saben mucho sobre cómo explicar conceptos técnicos y me enseñaron a enseñar. Escribí este libro para satisfacer una picazón particular: quería escribir un libro que explicara bien temas técnicos difíciles y quería un libro de algoritmos fácil de leer.

The first edition of this book came out in 2016. Since then, more than 100,000 people have read this book. I'm delighted to see how many people have connected with the visual learning style.

La primera edición de este libro se publicó en 2016. Desde entonces, más de 100.000 personas lo han leído. Estoy encantado de ver cuántas personas se han conectado con el estilo de aprendizaje visual.

With this second edition, my goal remains the same. In this book, I use illustrations and memorable examples to make concepts stick. The book is designed for readers who know

how to code and want to learn more about algorithms without any math knowledge required.

Con esta segunda edición, mi objetivo sigue siendo el mismo. En este libro, utilizo ilustraciones y ejemplos memorables para hacer que los conceptos se mantengan. El libro está diseñado para lectores que saben codificar y desean aprender más sobre algoritmos sin necesidad de conocimientos matemáticos.

The second edition fills some gaps in the first edition. I heard from a lot of readers that they wanted me to explain trees. There are now two chapters on trees in this book. I have also expanded the section on NP completeness. NP-complete is a very abstract concept, and I wanted an explanation that would make it more concrete. If you feel the same way, I hope the section on NP-complete fills that gap for you.

La segunda edición llena algunos vacíos de la primera edición. Muchos lectores me dijeron que querían que les explicara los árboles. Ahora hay dos capítulos sobre árboles en este libro. También he ampliado la sección sobre la integridad de NP. NP-completo es un concepto muy abstracto y quería una explicación que lo hiciera más concreto. Si siente lo mismo, espero que la sección sobre NP-completo llene ese vacío.

My writing has come a long way since that first blog post, and I hope you find this book an easy and informative read.

Mi escritura ha recorrido un largo camino desde esa primera publicación en el blog y espero que este libro le resulte fácil e informativo.

## ***acknowledgments***

### ***expresiones de gratitud***

Kudos to Manning for giving me the chance to write this book and letting me have a lot of creative freedom with it. Thanks to publisher Marjan Bace, Mike Stephens for getting me on board, and Ian Hough for being an incredibly responsive and helpful editor. Thanks also to the people on Manning's production team: Paul Wells, Debbie Holmgren, and all the others behind the scenes. In addition, I want to thank the many people who read the manuscript and offered suggestions: Daniel Zingaro, Ben Vinegar, Alexander Manning, and Maggie Wenger. Thanks to David Eisenstat, my technical reviewer, and Tony Holdroyd, the Manning technical proofreader, for catching my many errors.

Felicitaciones a Manning por darme la oportunidad de escribir este libro y permitirme tener mucha libertad creativa. Gracias a la editora Marjan Bace, a Mike Stephens por invitarme a participar y a Ian Hough por ser un editor increíblemente receptivo y útil. Gracias también a la gente del equipo de producción de Manning: Paul Wells, Debbie Holmgren y todos los demás detrás de escena. Además, quiero agradecer a las muchas personas que leyeron el manuscrito y ofrecieron sugerencias: Daniel Zingaro, Ben

Vinegar, Alexander Manning y Maggie Wenger. Gracias a David Eisenstat, mi revisor técnico, y a Tony Holdroyd, el corrector técnico de Manning, por detectar mis numerosos errores.

Thanks to the people who helped me reach this point: Bert Bates for teaching me how to write; the folks on the Flashkit game board for teaching me how to code; the many friends who helped by reviewing chapters, giving advice, and letting me try out different explanations, including Ben Vinegar, Karl Puzon, Alex Manning, Esther Chan, Anish Bhatt, Michael Glass, Nikrad Mahdi, Charles Lee, Jared Friedman, Hema Manickavasagam, Hari Raja, Murali Gudipati, Srinivas Varadan, and others, and Gerry Brady for teaching me algorithms. Another big thank you to algorithms academics like CLRS, Knuth, and Strang. I'm truly standing on the shoulders of giants.

Gracias a las personas que me ayudaron a llegar a este punto: Bert Bates por enseñarme a escribir; a la gente del tablero de juego Flashkit por enseñarme a codificar; los muchos amigos que me ayudaron revisando capítulos, dándome consejos y permitiéndome probar diferentes explicaciones, incluidos Ben Vinegar, Karl Puzon, Alex Manning, Esther Chan, Anish Bhatt, Michael Glass, Nikrad Mahdi, Charles Lee, Jared Friedman, Hema Manickavasagam., Hari Raja, Murali Gudipati, Srinivas Varadan y otros, y a Gerry Brady por enseñarme algoritmos. Otro gran agradecimiento a los académicos de algoritmos como CLRS, Knuth y Strang. Realmente estoy parado sobre los hombros de gigantes.

Dad, Mom, Priyanka, and the rest of the family: thank you for your constant support. And a big thank you to my wife Maggie, and my son Yogi. There are many adventures ahead of us, and some of them don't involve staying inside on a Friday night rewriting paragraphs.

Papá, mamá, Priyanka y el resto de la familia: gracias por su constante apoyo. Y muchas gracias a mi esposa Maggie y a mi hijo Yogi. Hay muchas aventuras por delante, y algunas de ellas no implican quedarnos adentro un viernes por la noche reescribiendo párrafos.

To all the reviewers—Abhishek Koserwal, Alex Lucas, Andres Sacco, Arun Saha, Becky Huett, Cesar Augusto Orozco Manotas, Christian Sutton, Diógenes Goldoni, Dirk Gómez, Ed Bacher, Eder Andres Avila Niño, Frans Oilinki, Ganesh Swaminathan, Giampiero Granatella, Glen Yu, Greg Kreiter, Javid Asgarov, João Ferreira, Jobinesh Purushothaman, Joe Cuevas, Josh McAdams, Krishna Anipindi, Krzysztof Kamyczek, Kyrylo Kalinichenko, Lakshminarayanan AS, Laud Bentil, Matteo Battista, Mikael Byström, Nick Rakochy, Ninoslav Cerkez, Oliver Korten, Ooi Kuan San, Pablo Varela, Patrick Regan, Patrick Wanjau, Philipp Konrad, Piotr Pindel, Rajesh Mohanan, Ranjit Sahai, Rohini Uppuluri, Roman Levchenko, Sambaran Hazra, Seth MacPherson, Shankar Swamy, Srihari Sridharan, Tobias Kopf, Vivek Veerappan, William Jamir Silva, and Xiangbo Mao—your suggestions helped make this a better book.

A todos los revisores: Abhishek Koserwal, Alex Lucas, Andres Sacco, Arun Saha, Becky Huett, Cesar Augusto

Orozco Manotas, Christian Sutton, Diógenes Goldoni, Dirk Gómez, Ed Bacher, Eder Andres Avila Niño, Frans Oilinki, Ganesh Swaminathan, Giampiero Granatella , Glen Yu, Greg Kreiter, Javid Asgarov, João Ferreira, Jobinesh Purushothaman, Joe Cuevas, Josh McAdams, Krishna Anipindi, Krzysztof Kamyczek, Kyrylo Kalinichenko, Lakshminarayanan AS, Laud Bentil, Matteo Battista, Mikael Byström, Nick Rakochy, Ninoslav Cerkez, Oliver Korten, Ooi Kuan San, Pablo Varela, Patrick Regan, Patrick Wanjau, Philipp Konrad, Piotr Pindel, Rajesh Mohanan, Ranjit Sahai, Rohini Uppuluri, Roman Levchenko, Sambaran Hazra, Seth MacPherson, Shankar Swamy, Srihari Sridharan, Tobias Kopf, Vivek Veerappan , William Jamir Silva y Xiangbo Mao: sus sugerencias ayudaron a hacer de este un libro mejor.

Finally, a big thank you to all the readers who took a chance on this book, and the readers who gave me feedback in the book's forum. You really helped make this book better.

Finalmente, muchas gracias a todos los lectores que se arriesgaron con este libro y a los lectores que me dieron su opinión en el foro del libro. Realmente ayudaste a mejorar este libro.

## ***about this book***

## ***sobre este libro***

*Grokking Algorithms* is designed to be easy to follow. I avoid big leaps of thought. Any time a new concept is

introduced, I explain it right away or tell you when I'll explain it. Core concepts are reinforced with exercises and multiple explanations so that you can check your assumptions and make sure you're following along.

Los algoritmos de Grokking están diseñados para ser fáciles de seguir. Evito grandes saltos de pensamiento. Cada vez que se introduce un concepto nuevo, lo explico de inmediato o les digo cuándo lo explicaré. Los conceptos básicos se refuerzan con ejercicios y múltiples explicaciones para que pueda comprobar sus suposiciones y asegurarse de seguirlas.

I lead with examples. Instead of writing symbol soup, my goal is to make it easy for you to visualize these concepts. I also think we learn best by being able to recall something we already know, and examples make recall easier. So when you're trying to remember the difference between arrays and linked lists (explained in chapter 2), you can just think about getting seated for a movie. Also, at the risk of stating the obvious, I'm a visual learner. This book is chock-full of images.

Lidero con ejemplos. En lugar de escribir sopa de símbolos, mi objetivo es facilitarle la visualización de estos conceptos. También creo que aprendemos mejor si podemos recordar algo que ya sabemos, y los ejemplos facilitan el recuerdo. Entonces, cuando intentes recordar la diferencia entre matrices y listas enlazadas (explicada en el capítulo 2), puedes pensar en sentarte para ver una película. Además, a

riesgo de decir lo obvio, soy un aprendiz visual. Este libro está repleto de imágenes.

The contents of the book are carefully curated. There's no need to write a book that covers every sorting algorithm—that's why we have Wikipedia and Khan Academy. All the algorithms I've included are practical. I've found them useful in my job as a software engineer, and they provide a good foundation for more complex topics. Happy reading!

El contenido del libro está cuidadosamente seleccionado. No es necesario escribir un libro que cubra todos los algoritmos de clasificación; por eso tenemos Wikipedia y Khan Academy. Todos los algoritmos que he incluido son prácticos. Los he encontrado útiles en mi trabajo como ingeniero de software y proporcionan una buena base para temas más complejos. ¡Feliz lectura!

## How to use this book

### Como usar este libro

The order and contents of this book have been carefully designed. If you're interested in a topic, feel free to jump ahead. Otherwise, read the chapters in order—they build on each other.

El orden y el contenido de este libro han sido cuidadosamente diseñados. Si está interesado en un tema, no dude en seguir adelante. De lo contrario, lea los capítulos en orden: se complementan unos con otros.

I strongly recommend executing the code for the examples yourself. I can't stress this part enough. Just type out my code samples verbatim (or download them from <https://www.manning.com/books/grokking-algorithms-second-edition> or [https://github.com/egonschiele/grokking\\_algorithms](https://github.com/egonschiele/grokking_algorithms)) and execute them. You'll retain a lot more if you do.

Recomiendo encarecidamente ejecutar el código de los ejemplos usted mismo. No puedo enfatizar lo suficiente esta parte. Simplemente escriba mis ejemplos de código palabra por palabra (o descárguelos de <https://www.manning.com/books/grokking-algorithms-first-edition> o [https://github.com/egonschiele/grokking\\_algorithms](https://github.com/egonschiele/grokking_algorithms)) y ejecútelos. Retendrás mucho más si lo haces.

I also recommend doing the exercises in this book. The exercises are short—usually just a minute or two, sometimes 5 to 10 minutes. They will help you check your thinking, so you'll know when you're off track before you've gone too far.

También recomiendo hacer los ejercicios de este libro. Los ejercicios son cortos: normalmente sólo uno o dos minutos, a veces de cinco a diez minutos. Le ayudarán a comprobar su forma de pensar, para que sepa cuándo se ha desviado antes de haber ido demasiado lejos.

## **Who should read this book?**

### **Quién debería leer este libro?**

This book is aimed at anyone who knows the basics of coding and wants to understand algorithms. Maybe you already have a coding problem and are trying to find an algorithmic solution. Or maybe you want to understand what algorithms are useful for. Here's a short, incomplete list of people who will probably find this book useful:

Este libro está dirigido a cualquier persona que conozca los conceptos básicos de codificación y quiera comprender los algoritmos. Quizás ya tengas un problema de codificación y estés intentando encontrar una solución algorítmica. O tal vez quieras entender para qué sirven los algoritmos. Aquí hay una lista breve e incompleta de personas que probablemente encontrarán útil este libro:

- Hobbyist coders  
Codificadores aficionados
- Coding boot camp students  
Estudiantes del campamento de entrenamiento de codificación
- Computer science grads looking for a refresher  
Graduados en informática que buscan un repaso

- Physics/math/other grads who are interested in programming
- Graduados en física/matemáticas/otros interesados en programación

## How this book is organized: A roadmap

### Cómo está organizado este libro: una hoja de ruta

The first three chapters of this book lay the foundations:

Los primeros tres capítulos de este libro sientan las bases:

- *Chapter 1*—You'll learn your first practical algorithm: binary search. You also learn to analyze the speed of an algorithm using big O notation. Big O notation is used throughout the book to analyze how slow or fast an algorithm is.

Capítulo 1: aprenderá su primer algoritmo práctico: la búsqueda binaria. También aprenderá a analizar la velocidad de un algoritmo utilizando la notación O grande. La notación O grande se utiliza a lo largo del libro para analizar qué tan lento o rápido es un algoritmo.

- *Chapter 2*—You'll learn about two fundamental data structures: arrays and linked lists. These data structures are used throughout the book, and they're used to make more advanced data structures like hash tables (chapter 5).

Capítulo 2: aprenderá sobre dos estructuras de datos fundamentales: matrices y listas vinculadas. Estas estructuras de datos se utilizan a lo largo del libro y se utilizan para crear estructuras de datos más avanzadas, como tablas hash (capítulo 5).

- *Chapter 3*—You'll learn about recursion, a handy technique used by many algorithms (such as quicksort, covered in chapter 4).

Capítulo 3: aprenderá sobre la recursividad, una técnica útil utilizada por muchos algoritmos (como la clasificación rápida, que se trata en el capítulo 4).

In my experience, big O notation and recursion are challenging topics for beginners. So I've slowed down and spent extra time on these sections.

En mi experiencia, la notación O grande y la recursividad son temas desafiantes para los principiantes. Así que reduje la velocidad y dediqué más tiempo a estas secciones.

The rest of the book presents algorithms with broad applications:

El resto del libro presenta algoritmos con amplias aplicaciones:

- *Problem-solving techniques*—Covered in chapters 4, 10, and 11. If you come across a problem and aren't sure how to solve it efficiently, try divide and conquer (chapter 4) or dynamic programming (chapter 11). Or you may realize there's no efficient solution, and get an approximate answer using a greedy algorithm instead (chapter 10).

Técnicas de resolución de problemas: cubiertas en los capítulos 4, 10 y 11. Si se encuentra con un problema y no está seguro de cómo resolverlo de manera eficiente, intente dividir y conquistar (capítulo 4) o programación dinámica (capítulo 11). O puede darse cuenta de que no existe una solución eficiente y obtener una respuesta aproximada utilizando un algoritmo codicioso (capítulo 10).

- *Hash tables*—Covered in chapter 5. A hash table is a very useful data structure. It contains sets of key and value pairs, like a person's name and their email address or a username and the associated password. It's hard to overstate hash tables' usefulness. When I want to solve a problem, the two plans of attack I start with are "Can I use a hash table?" and "Can I model this as a graph?"

Tablas hash: tratadas en el capítulo 5. Una tabla hash es una estructura de datos muy útil. Contiene conjuntos de pares de clave y valor, como el nombre de una persona y su dirección de correo electrónico o un nombre de usuario y la contraseña asociada. Es difícil exagerar la utilidad de las tablas hash. Cuando quiero resolver un problema, los dos planes de ataque con los que empiezo son "¿Puedo usar una tabla hash?" y "¿Puedo modelar esto como un gráfico?"

- *Graph and tree algorithms*—Covered in chapters 6, 7, 8, and 9. Graphs are a way to model a network: a social network, or a network of roads, or neurons, or any other set of connections. Breadth-first search (chapter 6) and Dijkstra's algorithm (chapter 9) are ways to find the shortest distance between two points in a network: you can use this approach to calculate the degrees of separation between two people or the shortest route to a destination. Trees are a type of graph. They are used in databases (often B-trees), in your browser (the DOM tree), or in your file system.

Algoritmos de gráficos y árboles: tratados en los capítulos 6, 7, 8 y 9. Los gráficos son una forma de modelar una red: una red social, una red de carreteras, neuronas o cualquier otro conjunto de conexiones. La búsqueda en amplitud (capítulo 6) y el algoritmo de Dijkstra (capítulo 9) son formas de encontrar la distancia más corta entre dos puntos en una red: puede utilizar este enfoque para calcular los grados de separación entre dos personas o la ruta más corta hacia un destino. . Los árboles son un tipo de gráfico. Se utilizan en bases de datos (a menudo árboles B), en su navegador (el árbol DOM) o en su sistema de archivos.

- *K-nearest neighbors (KNN)*—Covered in chapter 12. This is a simple machine-learning algorithm. You can use KNN to build a recommendations system, an OCR engine, a system to predict stock values—anything that involves predicting a value (“We think Adit will rate this movie 4 stars”) or classifying an object (“That letter is a Q”).

K vecinos más cercanos (KNN): se trata en el capítulo 12. Este es un algoritmo simple de aprendizaje automático. Puede utilizar KNN para crear un sistema de recomendaciones, un motor de OCR, un sistema para predecir valores de acciones, cualquier cosa que implique predecir un valor (“Creemos que Adit calificará esta película con 4 estrellas”) o clasificar un objeto (“Esa letra es una Q”).

- *Next steps*—Chapter 13 goes over more algorithms that would make good further reading.

Próximos pasos: el capítulo 13 analiza más algoritmos que serían una buena lectura adicional.

## About the code

### Sobre el código

All the code examples in this book use Python 3. All code in the book is presented in a fixed-width font like this to separate it from ordinary text. Code annotations accompany some of the listings, highlighting important concepts.

Todos los ejemplos de código de este libro utilizan Python 3. Todo el código del libro se presenta en un fixed-width font like this para separarlo del texto normal. Algunas de las listas están acompañadas de anotaciones de código que resaltan conceptos importantes.

You can get executable snippets of code from the liveBook (online) version of this book at

<https://livebook.manning.com/book/grokking-algorithms-second-edition>. The complete code for the examples in the book is available for download from the Manning website at [www.manning.com](http://www.manning.com) and from [https://github.com/egonschiele/grokking\\_algorithms](https://github.com/egonschiele/grokking_algorithms).

Puede obtener fragmentos de código ejecutables de la versión liveBook (en línea) de este libro en <https://livebook.manning.com/book/grokking-algorithms-first-edition>. El código completo de los ejemplos del libro está disponible para descargar desde el sitio web de Manning en [www.manning.com](http://www.manning.com) y desde [https://github.com/egonschiele/grokking\\_algorithms](https://github.com/egonschiele/grokking_algorithms).

I believe you learn best when you really enjoy learning—so have fun and run the code samples!

Creo que aprendes mejor cuando realmente disfrutas aprendiendo, iasí que divírtete y ejecuta los ejemplos de código!

## liveBook discussion forum

### Foro de discusión de LiveBook

Purchase of *Grokking Algorithms* includes free access to liveBook, Manning's online reading platform. Using liveBook's exclusive discussion features, you can attach comments to the book globally or to specific sections or paragraphs. It's a snap to make notes for yourself, ask and answer technical questions, and receive help from the author and other users. To access the forum, go to <https://livebook.manning.com/book/grokking-algorithms-second-edition>. You can also learn more about Manning's forums and the rules of conduct at <https://livebook.manning.com/discussion>.

La compra de Grokking Algorithms incluye acceso gratuito a liveBook, la plataforma de lectura en línea de Manning. Al utilizar las funciones de discusión exclusivas de LiveBook, puede adjuntar comentarios al libro de forma global o a secciones o párrafos específicos. Es muy fácil tomar notas, hacer y responder preguntas técnicas y recibir ayuda del autor y de otros usuarios. Para acceder al foro, vaya a <https://livebook.manning.com/book/grokking-algorithms-first-edition>. También puede obtener más información sobre los foros de Manning y las reglas de conducta en <https://livebook.manning.com/discussion>.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers

and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the author, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking the author some challenging questions lest their interest stray! The forum and the archives of previous discussions will be accessible from the publisher's website for as long as the book is in print.

El compromiso de Manning con nuestros lectores es proporcionar un lugar donde pueda tener lugar un diálogo significativo entre lectores individuales y entre lectores y el autor. No es un compromiso de participación específica por parte del autor, cuya contribución al foro sigue siendo voluntaria (y no remunerada). ¡Le sugerimos que intente hacerle al autor algunas preguntas desafiantes para que no se desvíe su interés! Se podrá acceder al foro y a los archivos de debates anteriores desde el sitio web del editor mientras el libro esté impreso.

## ***about the author***

### ***Sobre el Autor***



**ADITYA BHARGAVA** is a software engineer. He has a master's degree in computer science from the University of Chicago. He also runs a popular illustrated tech blog at adit.io.

Aditya Bhargava es ingeniero de software. Tiene una maestría en informática de la Universidad de Chicago. También dirige un popular blog de tecnología ilustrada en adit.io.

### **About the technical editor**

### **Sobre el editor técnico**

David Eisenstat is a research software engineer. He holds a PhD in Computer Science from Brown University.

David Eisenstat es ingeniero de software de investigación.  
Tiene un doctorado en Ciencias de la Computación de la  
Universidad de Brown.

# **1 Introduction to algorithms**

---

## **1 Introducción a los algoritmos**

---

### **In this chapter**

#### **En este capítulo**

- You get a foundation for the rest of the book.  
Obtienes una base para el resto del libro.
- You write your first search algorithm (binary search).  
Escribe su primer algoritmo de búsqueda (búsqueda binaria).
- You learn how to talk about the running time of an algorithm (big O notation).  
Aprenderá a hablar sobre el tiempo de ejecución de un algoritmo (notación O grande).

An *algorithm* is a set of instructions for accomplishing a task. Every piece of code could be called an algorithm, but this book covers the more interesting bits. I chose the algorithms in this book for inclusion because they're fast, or they solve interesting problems, or both. Here are some highlights:

Un algoritmo es un conjunto de instrucciones para realizar una tarea. Cada fragmento de código podría denominarse algoritmo, pero este libro cubre las partes más interesantes. Elegí los algoritmos de este libro para incluirlos porque son

rápidos, resuelven problemas interesantes, o ambas cosas. Aquí hay algunos aspectos destacados:

- Chapter 1 talks about binary search and shows how an algorithm can speed up your code. In one example, the number of steps needed goes from 4 billion down to 32!

El capítulo 1 habla sobre la búsqueda binaria y muestra cómo un algoritmo puede acelerar su código. En un ejemplo, el número de pasos necesarios pasa de 4 mil millones a 32!

- A GPS device uses graph algorithms (as you'll learn in

...

Un dispositivo GPS utiliza algoritmos gráficos (como aprenderá en...

## **2 Selection sort**

---

## **2 Orden de selección**

---

### **In this chapter**

### **En este capítulo**

- You learn about arrays and linked lists—two of the most basic data structures. They're used absolutely everywhere. You already used arrays in chapter 1, and you'll use them in almost every chapter in this book. Arrays are a crucial topic, so pay attention! But sometimes it's better to use a linked list instead of an array. This chapter explains the pros and cons of both so you can decide which one is right for your algorithm.

Aprenderá sobre matrices y listas vinculadas, dos de las estructuras de datos más básicas. Se utilizan absolutamente en todas partes. Ya usaste matrices en el capítulo 1 y las usarás en casi todos los capítulos de este libro. Las matrices son un tema crucial, ¡así que presta atención! Pero a veces es mejor utilizar una lista vinculada en lugar de una matriz. Este capítulo explica los pros y los contras de ambos para que pueda decidir cuál es el adecuado para su algoritmo.

- You learn your first sorting algorithm. A lot of algorithms only work if your data is sorted. Remember binary search? You can run binary search only on a sorted list of elements. This chapter teaches you selection sort. Most languages have a sorting algorithm built in, so you'll rarely need to write your own version from scratch. But selection sort is a stepping stone to quicksort, which I'll cover in chapter 4. Quicksort is an important algorithm, and it will be easier to understand if you know one sorting algorithm already.

Aprende su primer algoritmo de clasificación. Muchos algoritmos sólo funcionan si los datos están ordenados. ¿Recuerdas la búsqueda binaria? Puede ejecutar una búsqueda binaria solo en una lista ordenada de elementos. Este capítulo le enseña a ordenar por selección. La mayoría de los lenguajes tienen un algoritmo de clasificación incorporado, por lo que rara vez necesitarás escribir tu propia versión desde cero. Pero la clasificación por selección es un trampolín hacia la clasificación rápida, que cubriré en el capítulo 4. La clasificación rápida es un algoritmo importante y será más fácil de entender si ya conoce un algoritmo de clasificación.

## What you need to know

### Lo que necesitas saber

To understand the performance analysis bits in this chapter, you need to know big O notation and logarithms. If you don't know those, I suggest you go back and read chapter 1. Big O notation will be used throughout the rest of the book.

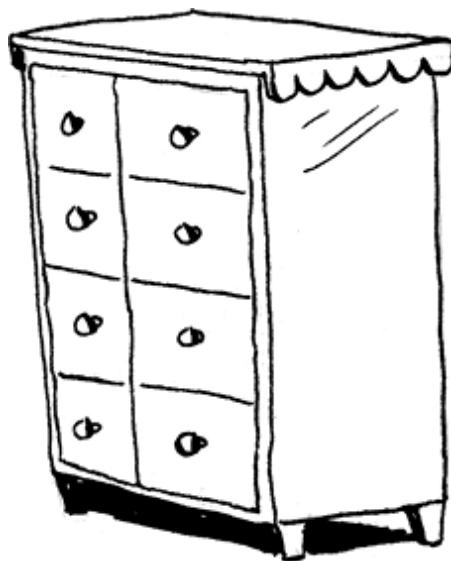
Para comprender los bits del análisis de rendimiento en este capítulo, necesita conocer la notación O grande y los logaritmos. Si no los conoce, le sugiero que regrese y lea el capítulo 1. La notación O grande se utilizará en el resto del libro.

## ***How memory works***

### ***como funciona la memoria***

Imagine you go to a show and need to check your things. A chest of drawers is available.

Imagina que vas a un espectáculo y necesitas revisar tus cosas. Una cómoda está disponible.



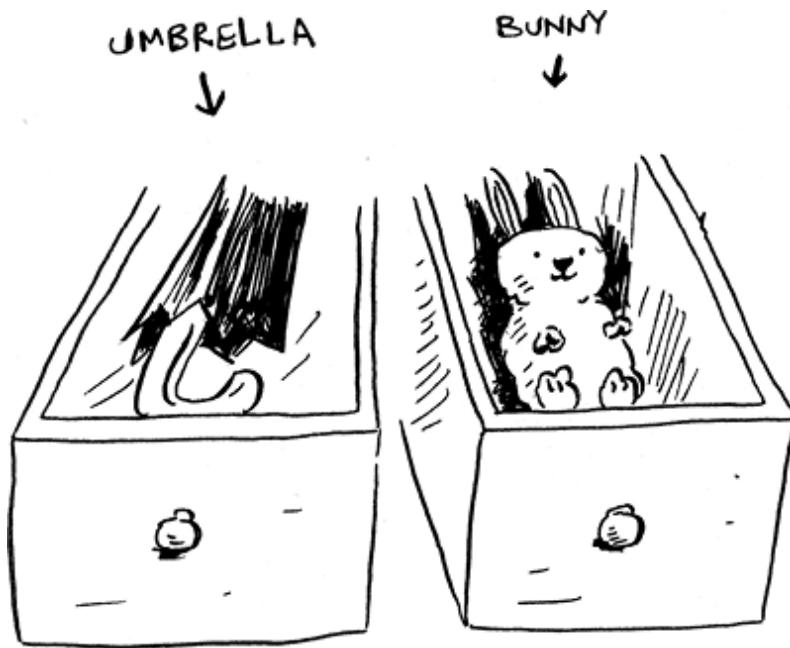
Each drawer can hold one element. You want to store two things, so you ask for two drawers.

Cada cajón puede contener un elemento. Quieres guardar dos cosas, entonces pides dos cajones.



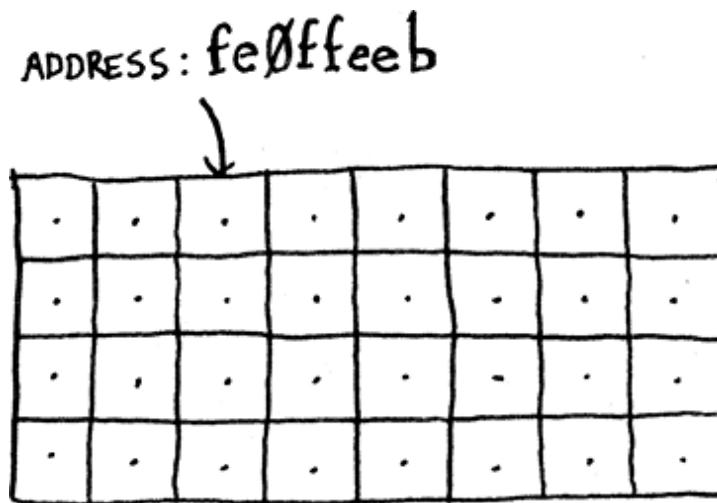
You store your two things here.

Guarda tus dos cosas aquí.



And you're ready for the show! This is basically how your computer's memory works. Your computer looks like a giant set of drawers, and each drawer has an address.

¡Y estás listo para el espectáculo! Básicamente, así es como funciona la memoria de su computadora. Su computadora parece un conjunto gigante de cajones y cada cajón tiene una dirección.



fe0ffe0b is the address of a slot in memory.

fe0ffe0b es la dirección de una ranura en la memoria.

Each time you want to store an item in memory, you ask the computer for some space, and it gives you an address where you can store your item. If you want to store multiple items, there are two basic ways to do so: arrays and linked lists. I'll talk about arrays and lists next, as well as the pros and cons of each. There isn't one right way to store items for every use case, so it's important to know the differences.

Cada vez que desea almacenar un elemento en la memoria, le solicita algo de espacio a la computadora y esta le proporciona una dirección donde puede almacenar su elemento. Si desea almacenar varios elementos, existen dos

formas básicas de hacerlo: matrices y listas vinculadas. A continuación hablaré de matrices y listas, así como de los pros y los contras de cada una. No existe una forma correcta de almacenar artículos para cada caso de uso, por lo que es importante conocer las diferencias.

## ***Arrays and linked lists***

### ***Matrices y listas enlazadas***

Sometimes you need to store a list of elements in memory. Suppose you're writing an app to manage your to-dos. You'll want to store the to-dos as a list in memory.

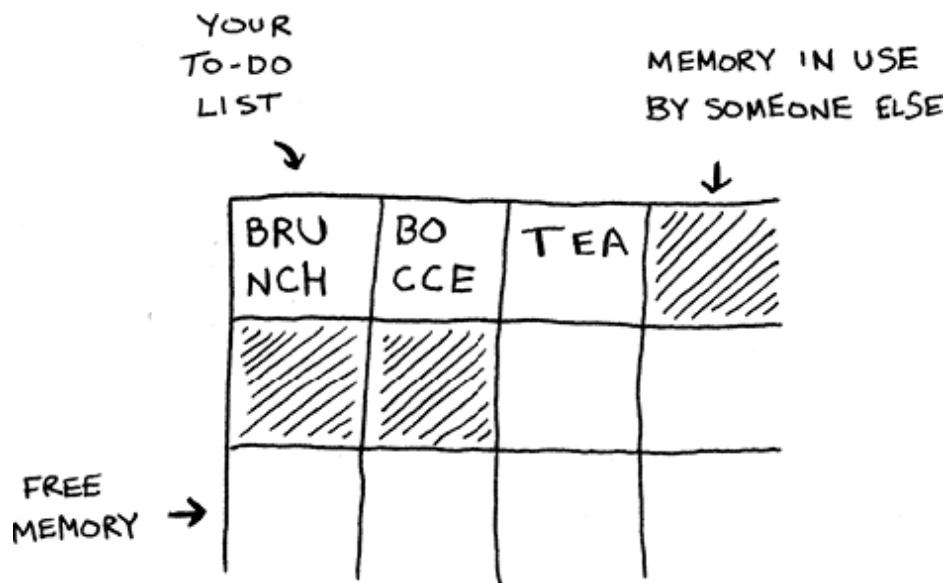
A veces es necesario almacenar una lista de elementos en la memoria. Suponga que está escribiendo una aplicación para administrar sus tareas pendientes. Querrá almacenar las tareas pendientes como una lista en la memoria.



Should you use an array or a linked list? Let's store the to-dos in an array first because it's easier to grasp. Using an

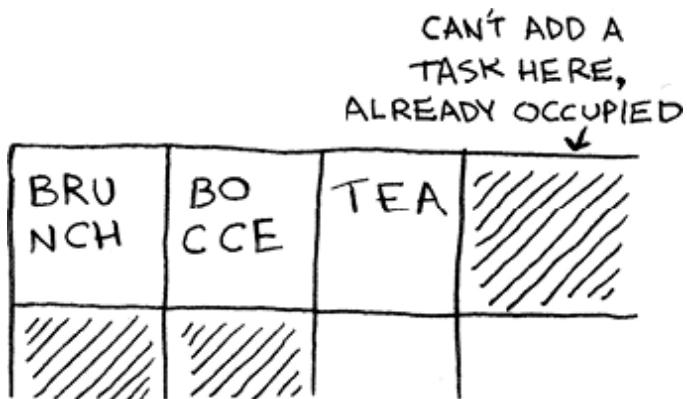
array means all your tasks are stored contiguously (right next to each other) in memory.

¿Debería utilizar una matriz o una lista vinculada? Primero almacenemos las tareas pendientes en una matriz porque es más fácil de entender. Usar una matriz significa que todas sus tareas se almacenan de forma contigua (una al lado de la otra) en la memoria.



Now, suppose you want to add a fourth task. But the next drawer is taken up by someone else's stuff!

Ahora supongamos que desea agregar una cuarta tarea. ¡Pero el siguiente cajón está ocupado por cosas de otra persona!



It's like going to a movie with your friends and finding a place to sit—but another friend joins you, and there's no place for them. You have to move to a new spot where you all fit. In this case, you need to ask your computer for a different chunk of memory that can fit four tasks. Then you need to move all your tasks there.

Es como ir al cine con tus amigos y encontrar un lugar para sentarse, pero otro amigo se une a ti y no hay lugar para ellos. Tienes que trasladarte a un nuevo lugar donde quepan todos. En este caso, deberá pedirle a su computadora una porción diferente de memoria que pueda acomodar cuatro tareas. Entonces necesitas mover todas tus tareas allí.

If another friend comes by, you're out of room again—and you all have to move a second time! What a pain. Similarly, adding new items to an array can be a big pain. If you're out of space and need to move to a new spot in memory every time, adding a new item will be really slow. One easy fix is to "hold seats": even if you have only three items in your task list, you can ask the computer for 10 slots, just in case. Then you can add up to 10 items to your task list without

having to move. This is a good workaround, but you should be aware of a couple of downsides:

Si viene otro amigo, os quedaréis sin espacio otra vez y itendréis que mudaros por segunda vez! Que dolor. De manera similar, agregar nuevos elementos a una matriz puede ser una gran molestia. Si se queda sin espacio y necesita moverse a un nuevo lugar en la memoria cada vez, agregar un nuevo elemento será muy lento. Una solución fácil es “reservar asientos”: incluso si solo tienes tres elementos en tu lista de tareas, puedes pedirle a la computadora 10 espacios, por si acaso. Luego podrás agregar hasta 10 elementos a tu lista de tareas sin tener que moverte. Esta es una buena solución, pero debes tener en cuenta un par de desventajas:

- You may not need the extra slots that you asked for, and then that memory will be wasted. You aren't using it, but no one else can use it either.

Es posible que no necesite las ranuras adicionales que solicitó y entonces esa memoria se desperdiciará. No lo estás usando, pero nadie más puede usarlo tampoco.

- You may add more than 10 items to your task list and have to move anyway.

Puede agregar más de 10 elementos a su lista de tareas y tener que moverse de todos modos.

So it's a good workaround, but it's not a perfect solution. Linked lists solve this problem of adding items.

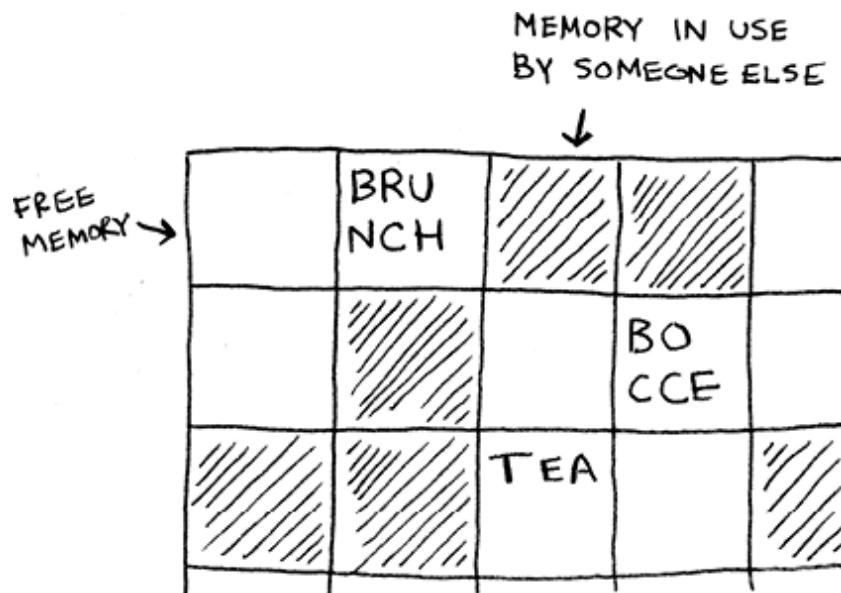
Por tanto, es una buena solución, pero no es una solución perfecta. Las listas enlazadas resuelven este problema de agregar elementos.

## Linked lists

### Listas enlazadas

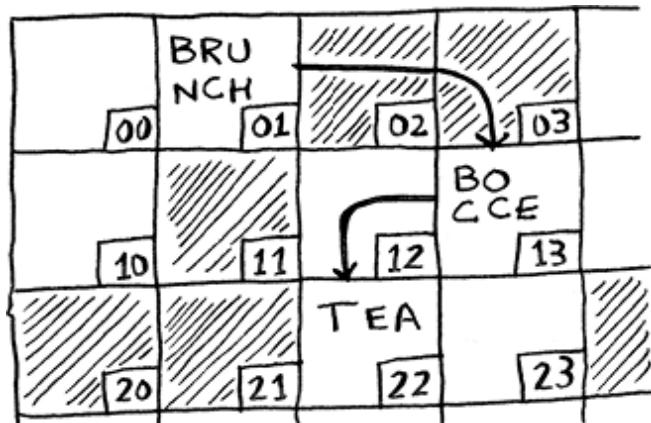
With linked lists, your items can be anywhere in memory.

Con las listas vinculadas, sus elementos pueden estar en cualquier lugar de la memoria.



Each item stores the address of the next item in the list. A bunch of random memory addresses are linked together.

Cada elemento almacena la dirección del siguiente elemento de la lista. Un montón de direcciones de memoria aleatorias están vinculadas entre sí.



### Linked memory addresses

#### Direcciones de memoria vinculadas

It's like a treasure hunt. You go to the first address, and it says, "The next item can be found at address 123." So you go to address 123, and it says, "The next item can be found at address 847," and so on. Adding an item to a linked list is easy: you stick it anywhere in memory and store the address with the previous item.

Es como una búsqueda del tesoro. Vas a la primera dirección y dice: "El siguiente artículo se puede encontrar en la dirección 123". Entonces vas a la dirección 123 y dice: "El siguiente artículo se puede encontrar en la dirección 847", y así sucesivamente. Agregar un elemento a una lista vinculada es fácil: lo pega en cualquier lugar de la memoria y almacena la dirección con el elemento anterior.

With linked lists, you never have to move your items. You also avoid another problem. Let's say you go to a popular movie with five of your friends. The six of you are trying to find a place to sit, but the theater is packed. There aren't six

seats together. Well, sometimes this happens with arrays. Let's say you're trying to find 10,000 slots for an array. Your memory has 10,000 slots, but it doesn't have 10,000 slots together. You can't get space for your array! A linked list is like saying, "Let's split up and watch the movie." If there's space in memory, you have space for your linked list.

Con las listas vinculadas, nunca tendrás que mover tus elementos. También evitas otro problema. Digamos que vas a ver una película popular con cinco de tus amigos. Ustedes seis están tratando de encontrar un lugar para sentarse, pero el teatro está lleno. No hay seis asientos juntos. Bueno, a veces esto sucede con las matrices. Digamos que estás intentando encontrar 10.000 espacios para una matriz. Su memoria tiene 10.000 espacios, pero no tiene 10.000 espacios juntos. ¡No puedes conseguir espacio para tu matriz! Una lista enlazada es como decir: "Separémonos y veamos la película". Si hay espacio en la memoria, tiene espacio para su lista vinculada.

If linked lists are so much better at inserts, what are arrays good for?

Si las listas enlazadas son mucho mejores en las inserciones, ¿para qué sirven las matrices?

## Arrays

### matrices

Websites with top-10 lists sometimes use this tactic to get more page views. Instead of showing you the list on one page, they put one item on each page and make you click Next to get to the next item in the list. For example, Top 10 Best TV Villains won't show you the entire list on one page. Instead, you start at #10 (Newman), and you have to click Next on each page to reach #1 (Gustavo Fring). This technique gives the websites 10 whole pages on which to show you ads, but it's boring to click Next nine times to get to #1. It would be much better if the whole list was on one page and you could click each person's name for more info.

Los sitios web con listas de los 10 principales a veces utilizan esta táctica para obtener más visitas a la página. En lugar de mostrarte la lista en una página, colocan un elemento en cada página y te hacen hacer clic en Siguiente para pasar al siguiente elemento de la lista. Por ejemplo, Los 10 mejores villanos de televisión no le mostrarán la lista completa en una sola página. En cambio, comienza en el puesto 10 (Newman) y debe hacer clic en Siguiente en cada página para llegar al puesto 1 (Gustavo Fring). Esta técnica proporciona a los sitios web 10 páginas completas para mostrarle anuncios, pero es aburrido hacer clic en Siguiente nueve veces para llegar al número 1. Sería mucho mejor si toda la lista estuviera en una sola página y pudieras hacer

clic en el nombre de cada persona para obtener más información.



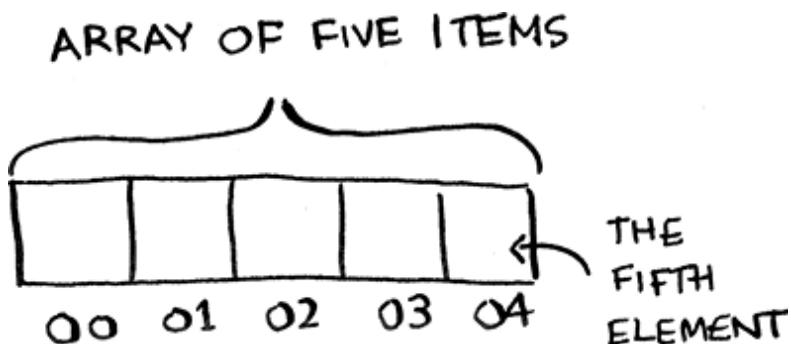
Linked lists have a similar problem. Suppose you want to read the last item in a linked list. You can't just read it because you don't know what address it's at. Instead, you have to go to item #1 to get the address for item #2. Then you have to go to item #2 to get the address for item #3. And so on, until you get to the last item. Linked lists are great if you're going to read all the items one at a time: you can read one item, follow the address to the next item, and so on. But if you're going to keep jumping around, linked lists are terrible.

Las listas enlazadas tienen un problema similar. Supongamos que desea leer el último elemento de una lista vinculada. No puedes simplemente leerlo porque no sabes en qué dirección está. En su lugar, debe ir al artículo n.º 1 para obtener la dirección del artículo n.º 2. Luego debe ir al artículo n.º 2 para obtener la dirección del artículo n.º 3. Y así sucesivamente hasta llegar al último elemento. Las listas enlazadas son excelentes si va a leer todos los elementos uno a la vez: puede leer un elemento, seguir la dirección

hasta el siguiente, etc. Pero si vas a seguir saltando, las listas enlazadas son terribles.

Arrays are different. You know the address for every item in your array. For example, suppose your array contains five items, and you know it starts at address 00. What is the address of item #5?

Las matrices son diferentes. Conoce la dirección de cada elemento de su matriz. Por ejemplo, suponga que su matriz contiene cinco elementos y sabe que comienza en la dirección 00. ¿Cuál es la dirección del elemento n.º 5?



Simple math tells you: it's 04. Arrays are great if you want to read random elements because you can look up any element in your array instantly. With a linked list, the elements aren't next to each other, so you can't instantly calculate the position of the fifth element in memory—you have to go to the first element to get the address to the second element, then go to the second element to get the address of the third element, and so on until you get to the fifth element.

Las matemáticas simples te dicen: es 04. Las matrices son excelentes si quieras leer elementos aleatorios porque puedes buscar cualquier elemento en tu matriz al instante. Con una lista enlazada, los elementos no están uno al lado del otro, por lo que no puedes calcular instantáneamente la posición del quinto elemento en la memoria; tienes que ir al primer elemento para obtener la dirección del segundo elemento, luego ir al segundo elemento para obtener la dirección del tercer elemento, y así sucesivamente hasta llegar al quinto elemento.

## Terminology

### Terminología

The elements in an array are numbered. This numbering starts from 0, not 1. For example, in this array, 20 is at position 1.

Los elementos de una matriz están numerados. Esta numeración comienza desde 0, no desde 1. Por ejemplo, en esta matriz, 20 está en la posición 1.

10	20	30	40
∅	1	2	3

And 10 is at position 0. This usually throws new programmers for a spin. Starting at 0 makes all kinds of array-based code easier to write, so programmers have

stuck with it. Almost every programming language you use will number array elements starting at 0. You'll soon get used to it.

Y 10 está en la posición 0. Esto generalmente hace que los nuevos programadores prueben. Comenzar en 0 hace que todo tipo de código basado en matrices sea más fácil de escribir, por lo que los programadores se han quedado con él. Casi todos los lenguajes de programación que utilice numerarán elementos de matriz comenzando en 0. Pronto se acostumbrará.

The position of an element is called its *index*. So instead of saying, "20 is at *position 1*," the correct terminology is, "20 is at *index 1*." I'll use *index* to mean *position* throughout this book.

La posición de un elemento se llama índice. Entonces, en lugar de decir "20 está en la posición 1", la terminología correcta es "20 está en el índice 1". Usaré índice para referirme a posición a lo largo de este libro.

Here are the run times for common operations on arrays and lists.

Estos son los tiempos de ejecución para operaciones comunes en matrices y listas.

	ARRAYS	LISTS
READING	$O(1)$	$O(n)$
INSERTION	$O(n)$	$O(1)$

$O(n)$  = LINEAR TIME  
 $O(1)$  = CONSTANT TIME

Question: Why does it take  $O(n)$  time to insert an element into an array? Suppose you wanted to insert an element at the beginning of an array. How would you do it? How long would it take? Find the answers to these questions in the next section!

Pregunta: ¿Por qué lleva  $O(n)$  tiempo insertar un elemento en una matriz? Supongamos que desea insertar un elemento al comienzo de una matriz. ¿Como lo harías? ¿Cuanto tiempo tardaría? ¡Encuentre las respuestas a estas preguntas en la siguiente sección!

## EXERCISE

### EJERCICIO

**2.1** Suppose you're building an app to keep track of your finances.

2.1 Suponga que está creando una aplicación para realizar un seguimiento de sus finanzas.

- 1. GROCERIES**
- 2. MOVIE**
- 3. SFBC  
MEMBERSHIP**

Every day, you write down everything you spent money on. At the end of the month, you review your expenses and sum up how much you spent. So you have lots of inserts and a few reads. Should you use an array or a list?

Todos los días, anotas todo en lo que gastaste dinero. Al final del mes, revisas tus gastos y sumas cuánto gastaste. Entonces tienes muchas inserciones y algunas lecturas. ¿Deberías usar una matriz o una lista?

## **Inserting into the middle of a list**

### **Insertar en medio de una lista**

Suppose you want your to-do list to work more like a calendar. Earlier, you were adding things to the end of the list.

Suponga que desea que su lista de tareas pendientes funcione más como un calendario. Antes, agregabas cosas al final de la lista.

Now, you want to add them in the order in which they should be done.

Ahora desea agregarlos en el orden en que deben hacerse.



**Unordered**

**desordenado**



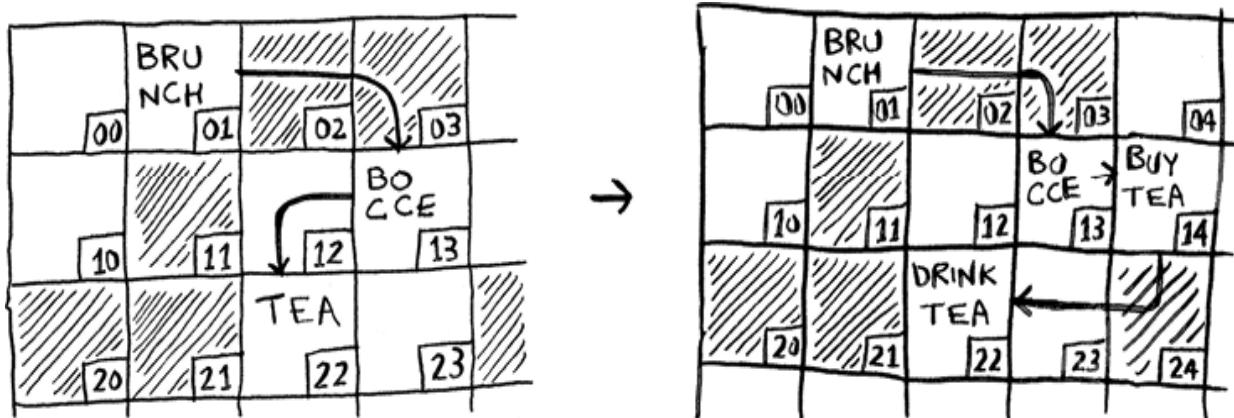
**Ordered**

**Ordenado**

What's better if you want to insert elements in the middle: arrays or lists? With lists, it's as easy as changing what the previous element points to.

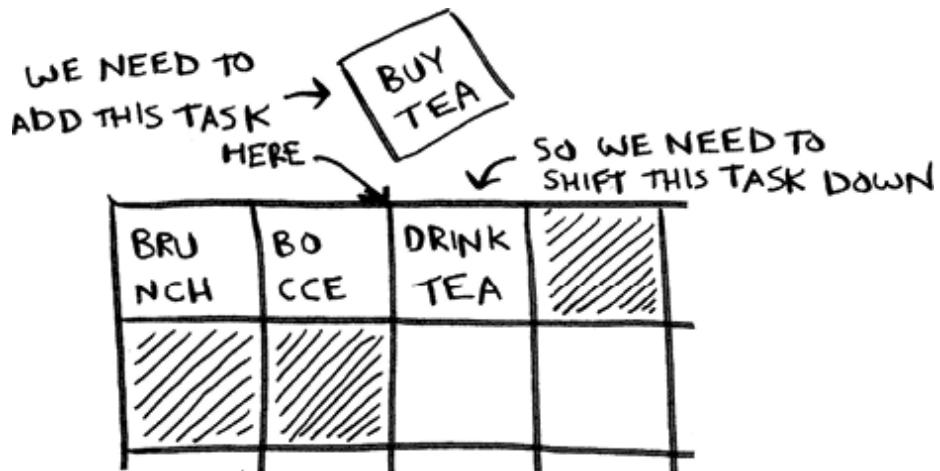
¿Qué es mejor si quieres insertar elementos en el medio: matrices o listas? Con las listas es tan fácil como cambiar a

qué apunta el elemento anterior.



But for arrays, you have to shift all the rest of the elements down.

Pero para las matrices, debes desplazar el resto de los elementos hacia abajo.



And if there's no space, you might have to copy everything to a new location! Lists are better if you want to insert elements into the middle.

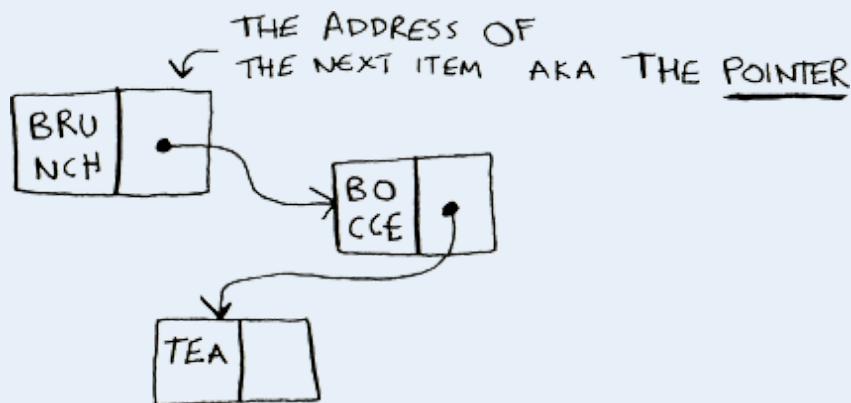
Y si no hay espacio, es posible que tengas que copiar todo en una nueva ubicación! Las listas son mejores si deseas insertar elementos en el medio.

## Pointers

### Consejos

I have talked a lot about how each item in a linked list points to the next item in the list. But how does it do that exactly? By using pointers.

He hablado mucho sobre cómo cada elemento de una lista vinculada apunta al siguiente elemento de la lista. ¿Pero cómo hace eso exactamente? Mediante el uso de punteros.



With each item in your linked list, you use a little bit of memory to store the address of the next item. This is called a *pointer*.

Con cada elemento de su lista vinculada, utiliza un poco de memoria para almacenar la dirección del siguiente elemento. Esto se llama puntero.

You will hear the word *pointers* come up sometimes, especially if you write using a lower-level language like C. So it's good to know what it means.

A veces oirás aparecer las palabras punteros, especialmente si escribes utilizando un lenguaje de nivel inferior como C. Por eso es bueno saber lo que significa.

## Deletions

### Eliminaciones

What if you want to delete an element? Again, lists are better because you just need to change what the previous element points to. With arrays, everything needs to be moved up when you delete an element.

¿Qué pasa si quieres eliminar un elemento? Nuevamente, las listas son mejores porque solo necesitas cambiar lo que apunta el elemento anterior. Con las matrices, es necesario mover todo hacia arriba cuando eliminas un elemento.

Unlike insertions, deletions will always work. Insertions can fail sometimes when there's no space left in memory. But you can always delete an element.

A diferencia de las inserciones, las eliminaciones siempre funcionarán. A veces, las inserciones pueden fallar cuando no queda espacio en la memoria. Pero siempre puedes eliminar un elemento.

Here are the run times for common operations on arrays and linked lists.

Estos son los tiempos de ejecución para operaciones comunes en matrices y listas vinculadas.

	ARRAYS	LISTS
READING	O(1)	O(n)
INSERTION	O(n)	O(1)
DELETION	O(n)	O(1)

It's worth mentioning that insertions and deletions are O(1) time only if you can instantly access the element to be deleted. It's a common practice to keep track of the first and last items in a linked list, so it would take only O(1) time to delete those.

Vale la pena mencionar que las inserciones y eliminaciones son de tiempo O(1) solo si puede acceder instantáneamente al elemento que desea eliminar. Es una práctica común realizar un seguimiento del primer y último elemento de una lista vinculada, por lo que solo tomaría O(1) tiempo eliminarlos.

## ***Which is used more, arrays or linked lists?***

## ***¿Qué se utiliza más, los arrays o las listas enlazadas?***

Arrays are often used because they have a lot of advantages over linked lists. First, they are better at reads. Arrays provide random access.

Las matrices se utilizan a menudo porque tienen muchas ventajas sobre las listas enlazadas. Primero, son mejores leyendo. Las matrices proporcionan acceso aleatorio.

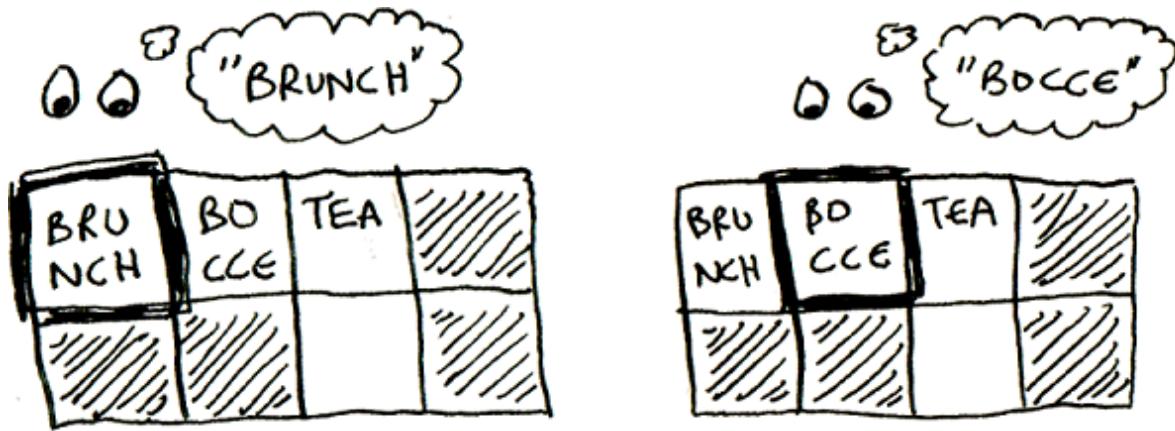
There are two different types of access: random access and sequential access. Sequential access means reading the elements one by one, starting with the first element. Linked lists can only do sequential access. If you want to read the 10th element of a linked list, you have to read the first nine elements and follow the links to the 10th element. Random access means you can jump directly to the 10th element. Arrays provide random access. A lot of use cases require random access, so arrays are used a lot.

Existen dos tipos diferentes de acceso: acceso aleatorio y acceso secuencial. El acceso secuencial significa leer los elementos uno por uno, comenzando con el primer elemento. Las listas enlazadas sólo pueden realizar acceso secuencial. Si desea leer el décimo elemento de una lista vinculada, debe leer los primeros nueve elementos y seguir los enlaces al décimo elemento. El acceso aleatorio significa que puedes saltar directamente al décimo elemento. Las matrices proporcionan acceso aleatorio. Muchos casos de uso requieren acceso aleatorio, por lo que se utilizan mucho las matrices.

Even beyond random access, though, arrays are faster because they can use caching. Maybe you are picturing reads like this, reading one item at a time.

Sin embargo, incluso más allá del acceso aleatorio, las matrices son más rápidas porque pueden utilizar el

almacenamiento en caché. Tal vez te estés imaginando lecturas como ésta, leyendo un elemento a la vez.



But in reality, computers read a whole section at a time because that makes it a lot faster to go to the next item:

Pero en realidad, las computadoras leen una sección completa a la vez porque eso hace que sea mucho más rápido pasar al siguiente elemento:



This is something you can do with arrays. With an array, you can read a whole section of items. But you can't do this with a linked list! You don't know where the next item is. You

need to read an item, find out where the next item is, and then read the next item.

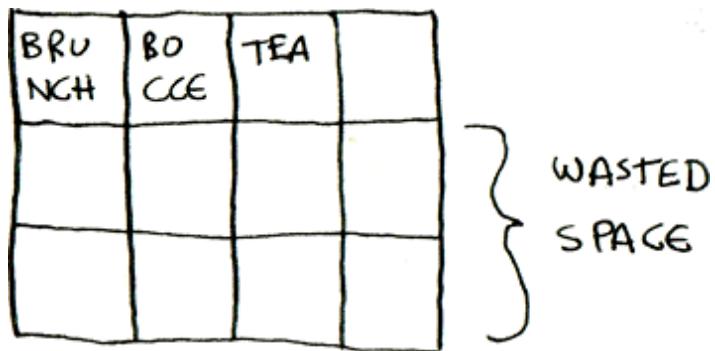
Esto es algo que puedes hacer con matrices. Con una matriz, puedes leer una sección completa de elementos. ¡Pero no puedes hacer esto con una lista vinculada! No sabes dónde está el siguiente elemento. Necesita leer un elemento, averiguar dónde está el siguiente y luego leer el siguiente.

So not only do arrays give you random access, but they also provide faster sequential access!

Entonces, las matrices no solo le brindan acceso aleatorio, sino que también brindan un acceso secuencial más rápido.

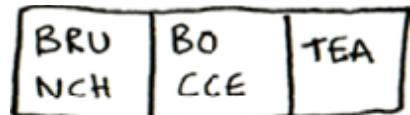
Arrays are better for reads. What about memory efficiency? Remember earlier I said that with arrays, you typically request more space than you need, and if you don't end up using that extra memory you requested, it is wasted?

Las matrices son mejores para lecturas. ¿Qué pasa con la eficiencia de la memoria? ¿Recuerda que antes dije que con las matrices, normalmente solicita más espacio del que necesita y, si no termina usando esa memoria adicional que solicitó, se desperdicia?

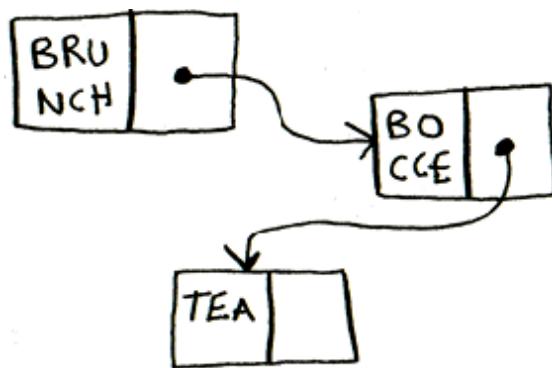


Well, in reality, there is not much wasted space like this. On the other hand, when you use a linked list, you are using extra memory per item because you need to store the address of the next item. So linked lists will take up more space if each item is pretty small. Here's the same information as an array and a linked list. You can see the linked list takes up more space.

Bueno, en realidad, no hay mucho espacio desperdiciado como este. Por otro lado, cuando usa una lista vinculada, está usando memoria adicional por elemento porque necesita almacenar la dirección del siguiente elemento. Por tanto, las listas enlazadas ocuparán más espacio si cada elemento es bastante pequeño. Aquí se muestra la misma información que una matriz y una lista vinculada. Puede ver que la lista vinculada ocupa más espacio.



ARRAYS



LINKED LISTS

Of course, if each item is big, then even a single slot of wasted space can be a big deal, and that extra memory you're using to store the pointers can feel pretty small by comparison.

Por supuesto, si cada elemento es grande, entonces incluso una sola ranura de espacio desperdiciado puede ser un gran problema, y esa memoria adicional que estás usando para almacenar los punteros puede parecer bastante pequeña en comparación.

So arrays are used more often than linked lists except in specific use cases.

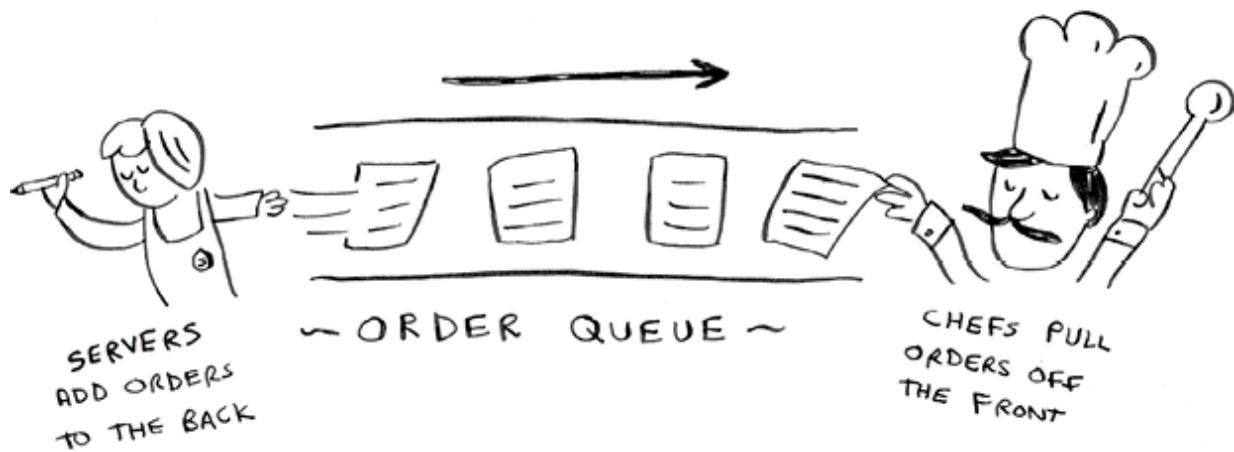
Por lo tanto, las matrices se utilizan con más frecuencia que las listas vinculadas, excepto en casos de uso específicos.

## EXERCISES

### EJERCICIOS

**2.2** Suppose you're building an app for restaurants to take customer orders. Your app needs to store a list of orders. Servers keep adding orders to this list, and chefs take orders off the list and make them. It's an order queue: servers add orders to the back of the queue, and the chef takes the first order off the queue and cooks it.

2.2 Suponga que está creando una aplicación para que los restaurantes tomen pedidos de los clientes. Tu aplicación necesita almacenar una lista de pedidos. Los camareros siguen agregando pedidos a esta lista y los chefs quitan pedidos de la lista y los preparan. Es una cola de pedidos: los servidores agregan pedidos al final de la cola y el chef toma el primer pedido de la cola y lo cocina.



Would you use an array or a linked list to implement this queue? (Hint: Linked lists are good for inserts/deletes, and arrays are good for random access. Which one are you going to be doing here?)

¿Usarías una matriz o una lista vinculada para implementar esta cola? (Sugerencia: las listas vinculadas son buenas para insertar/eliminar, y las matrices son buenas para el acceso aleatorio. ¿Cuál vas a hacer aquí?)

**2.3** Let's run a thought experiment. Suppose Facebook keeps a list of usernames. When someone tries to log in to Facebook, a search is done for their username. If their name is in the list of usernames, they can log in. People log in to Facebook pretty often, so there are a lot of searches through this list of usernames. Suppose Facebook uses binary search to search the list. Binary search needs random access—you need to be able to get to the middle of

the list of usernames instantly. Knowing this, would you implement the list as an array or a linked list?

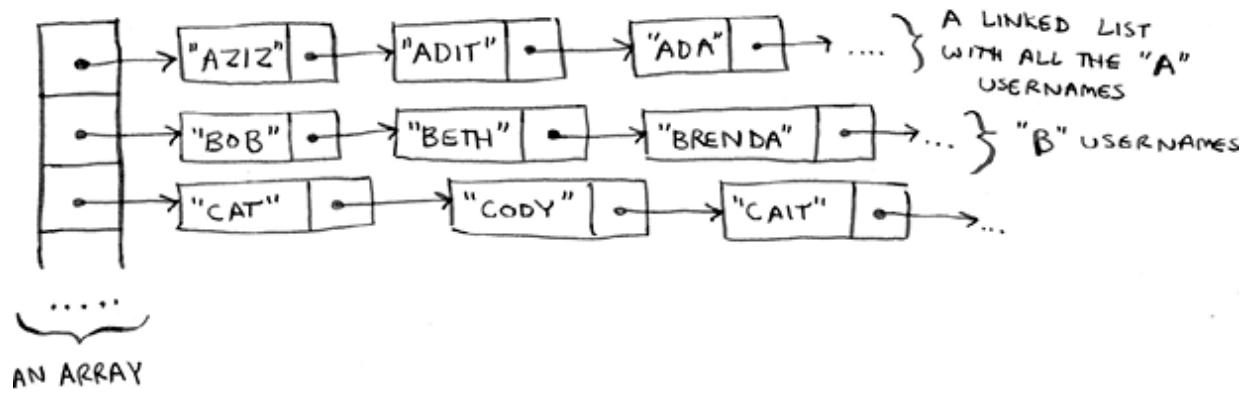
2.3 Hagamos un experimento mental. Supongamos que Facebook mantiene una lista de nombres de usuario. Cuando alguien intenta iniciar sesión en Facebook, se realiza una búsqueda de su nombre de usuario. Si su nombre está en la lista de nombres de usuario, pueden iniciar sesión. La gente inicia sesión en Facebook con bastante frecuencia, por lo que hay muchas búsquedas en esta lista de nombres de usuario. Supongamos que Facebook usa la búsqueda binaria para buscar en la lista. La búsqueda binaria necesita acceso aleatorio: debe poder llegar al centro de la lista de nombres de usuario al instante. Sabiendo esto, ¿implementarías la lista como una matriz o una lista enlazada?

**2.4** People sign up for Facebook pretty often, too. Suppose you decided to use an array to store the list of users. What are the downsides of an array for inserts? In particular, suppose you're using binary search to search for logins. What happens when you add new users to an array?

2.4 La gente también se registra en Facebook con bastante frecuencia. Suponga que decide utilizar una matriz para almacenar la lista de usuarios. ¿Cuáles son las desventajas de una matriz para inserciones? En particular, supongamos que está utilizando la búsqueda binaria para buscar inicio de sesión. ¿Qué sucede cuando agrega nuevos usuarios a una matriz?

**2.5** In reality, Facebook uses neither an array nor a linked list to store user information. Let's consider a hybrid data structure: an array of linked lists. You have an array with 26 slots. Each slot points to a linked list. For example, the first slot in the array points to a linked list containing all the usernames starting with *A*. The second slot points to a linked list containing all the usernames starting with *B*, and so on.

2.5 En realidad, Facebook no utiliza ni una matriz ni una lista enlazada para almacenar información del usuario. Consideremos una estructura de datos híbrida: una serie de listas enlazadas. Tienes una matriz con 26 ranuras. Cada ranura apunta a una lista vinculada. Por ejemplo, el primer espacio de la matriz apunta a una lista vinculada que contiene todos los nombres de usuario que comienzan con *A*. El segundo espacio apunta a una lista vinculada que contiene todos los nombres de usuario que comienzan con *B*, y así sucesivamente.



Suppose Adit B signs up for Facebook, and you want to add them to the list. You go to slot 1 in the array, go to the linked list for slot 1, and add Adit B at the end. Now, suppose you want to search for Zakhir H. You go to slot 26, which points to a linked list of all the Z names. Then you search through that list to find Zakhir H. Compare this hybrid data structure to arrays and linked lists. Is it slower or faster than each for searching and inserting? You don't have to give big O run times, just whether the data structure would be faster or slower.

Supongamos que Adit B se registra en Facebook y desea agregarlo a la lista. Vaya a la ranura 1 de la matriz, vaya a la lista vinculada para la ranura 1 y agregue Adit B al final. Ahora, supongamos que desea buscar a Zakhir H. Vaya al espacio 26, que apunta a una lista vinculada de todos los nombres Z. Luego busca en esa lista para encontrar a Zakhir H. Compare esta estructura de datos híbrida con matrices y listas vinculadas. ¿Es más lento o más rápido que cada uno para buscar e insertar? No es necesario indicar

tiempos de ejecución O grandes, solo si la estructura de datos sería más rápida o más lenta.

## ***Selection sort***

### ***Orden de selección***

Let's put it all together to learn your second algorithm: selection sort. To follow this section, you need to understand arrays and big O notation.

Juntémoslo todo para aprender su segundo algoritmo: ordenación por selección. Para seguir esta sección, debes comprender las matrices y la notación O grande.



Suppose you have a bunch of music on your computer. For each artist, you have a play count.

Suponga que tiene un montón de música en su computadora. Para cada artista, tienes un recuento de reproducciones.

~ ♫ ~		PLAY COUNT
RADIOHEAD		156
KISHORE KUMAR		141
THE BLACK KEYS		35
NEUTRAL MILK HOTEL		94
BECK		88
THE STROKES		61
WILCO		111

You want to sort these artists from most to least played so that you can rank your favorite artists. How can you do it?

Desea ordenar estos artistas del más al menos reproducido para poder clasificar a sus artistas favoritos. ¿Cómo puedes hacerlo?

One way is to go through the list and find the most-played artist. Add that artist to a new list.

Una forma es revisar la lista y encontrar el artista más reproducido. Añade ese artista a una nueva lista.

~ ♫ ~

	PLAY COUNT
RADIOHEAD	156
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111



SORTED ↘

	PLAY COUNT
RADIOHEAD	156

1. RADIOHEAD  
IS THE MOST PLAYED ARTIST...

2. ADD IT TO A NEW LIST

Do it again to find the next-most-played artist.

Hazlo de nuevo para encontrar el siguiente artista más reproducido.

♪ ♪ ♪	PLAY COUNT
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111

→

♪ SORTED ♪	PLAY COUNT
RADIOHEAD	156
KISHORE KUMAR	141

1. KISHORE KUMAR  
IS THE NEXT  
MUST-PLAYED  
ARTIST

2. SO IT IS  
THE NEXT ARTIST  
ADDED TO THE  
NEW LIST

Keep doing this, and you'll end up with a sorted list.

Continúe haciendo esto y terminará con una lista ordenada.

	PLAY COUNT
RADIOHEAD	156
KISHORE KUMAR	141
WILCO	111
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
THE BLACK KEYS	35

Let's put on our computer science hats and see how long this will take to run. Remember that  $O(n)$  time means you touch every element in a list once. For example, running a simple search over the list of artists means looking at each artist once.

Pongámonos el sombrero de informática y veamos cuánto tiempo tardará en ejecutarse. Recuerde que  $O(n)$  time significa que toca cada elemento de una lista una vez. Por ejemplo, realizar una búsqueda simple en la lista de artistas significa mirar a cada artista una vez.

- 1. RADIOHEAD
  - 2. KISHORE KUMAR
  - 3. THE BLACK KEYS
  - 4. NEUTRAL MILK HOTEL
  - 5. BECK
  - 6. THE STROKES
  - 7. WILCO
- 
- n  
ITEMS

To find the artist with the highest play count, you have to check each item in the list. This takes  $O(n)$  time, as you just saw. So you have an operation that takes  $O(n)$  time, and you have to do that  $n$  times.

Para encontrar el artista con el mayor número de reproducciones, debes marcar cada elemento de la lista. Esto lleva  $O(n)$  tiempo, como acaba de ver. Entonces tienes una operación que toma  $O(n)$  tiempo y tienes que hacerla  $n$  veces.

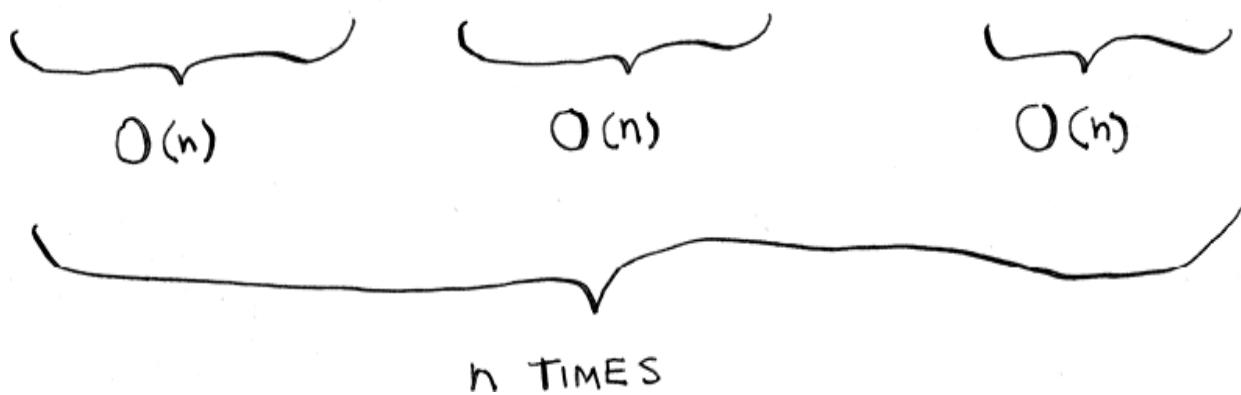
1. RADIOHEAD
2. KISHORE KUMAR
3. THE BLACK KEYS
4. NEUTRAL MILK HOTEL
5. BECK
6. THE STROKES
7. WILCO



1. KISHORE KUMAR
2. THE BLACK KEYS
3. NEUTRAL MILK HOTEL
4. BECK
5. THE STROKES
6. WILCO

...

1. THE STROKES



This takes  $O(n \times n)$  time or  $O(n^2)$  time.

Esto lleva  $O(n \times n)$  tiempo o  $O(n^2)$  tiempo.

Sorting algorithms are very useful. Now you can sort

Los algoritmos de clasificación son muy útiles. Ahora puedes ordenar

- Names in a phone book  
Nombres en una guía telefónica
- Travel dates  
Fechas de viaje
- Emails (newest to oldest)  
Correos electrónicos (del más nuevo al más antiguo)

## Checking fewer elements each time

### Comprobando menos elementos cada vez

As you go through the operations, the number of elements you have to check keeps decreasing. Eventually, you're down to having to check just one element. So maybe you are wondering: How can the run time still be  $O(n^2)$ ? That's a good question, and the answer has to do with constants in big O notation. I'll get into this more in chapter 4, but here's the gist.

A medida que avanzas en las operaciones, la cantidad de elementos que debes verificar sigue disminuyendo. Al final, tendrás que comprobar solo un elemento. Quizás te estés preguntando: ¿Cómo es posible que el tiempo de ejecución siga siendo  $O(n^2)$ ? Ésa es una buena pregunta y la respuesta tiene que ver con las constantes en notación O grande. Hablaré más de esto en el capítulo 4, pero aquí está la esencia.

You're right that you don't have to check a list of  $n$  elements each time. You check  $n$  elements, then  $n - 1$ ,  $n - 2$ , ..., 2, 1. On average, you check a list that has  $1/2 \times n$  elements. The runtime is  $O(n \times 1/2 \times n)$ . But constants like  $1/2$  are ignored in big O notation (again, see chapter 4 for the full discussion), so you just write  $O(n \times n)$  or  $O(n^2)$ .

Tienes razón en que no es necesario comprobar una lista de  $n$  elementos cada vez. Compruebas  $n$  elementos, luego  $n - 1$ ,  $n - 2$ , ..., 2, 1. En promedio, verifica una lista que tiene  $1/2 \times n$  elementos. El tiempo de ejecución es  $O(n \times 1/2 \times n)$ . Pero las constantes como  $1/2$  se ignoran en la notación O grande (de nuevo, consulte el capítulo 4 para ver la discusión completa), por lo que simplemente escribe  $O(n \times n)$  u  $O(n^2)$ .

Selection sort is a neat algorithm, but it's not very fast. Quicksort is a faster sorting algorithm that only takes  $O(n \log n)$  time. It's coming up in chapter 4!

La clasificación por selección es un algoritmo sencillo, pero no muy rápido. Quicksort es un algoritmo de clasificación más rápido que solo requiere  $O(n \log n)$  tiempo. ¡Viene en el capítulo 4!

## **Example code listing**

### **Listado de códigos de ejemplo**

I didn't show you the code to sort the music list, but the following is some code that will do something very similar: sort an array from smallest to largest. Let's write a function to find the smallest element in an array:

No te mostré el código para ordenar la lista de música, pero el siguiente es un código que hará algo muy similar: ordenar una matriz de menor a mayor. Escribamos una función para encontrar el elemento más pequeño en una matriz:

```
def findSmallest(arr):
    smallest = arr[0]          ①
    smallest_index = 0          ②
    for i in range(1, len(arr)):
        if arr[i] < smallest:
            smallest = arr[i]
            smallest_index = i
    return smallest_index
```

- ① Stores the smallest value
- ① Almacena el valor más pequeño
- ② Stores the index of the smallest value
- ② Almacena el índice del valor más pequeño.

Now you can use this function to write selection sort:

Ahora puedes usar esta función para escribir clasificación de selección:

```
def selectionSort(arr):          ①
    newArr = []
```

```
copiedArr = list(arr) // copy array before mutating
for i in range(len(copiedArr)):
    smallest = findSmallest(copiedArr)      ②
    newArr.append(copiedArr.pop(smallest))
return newArr

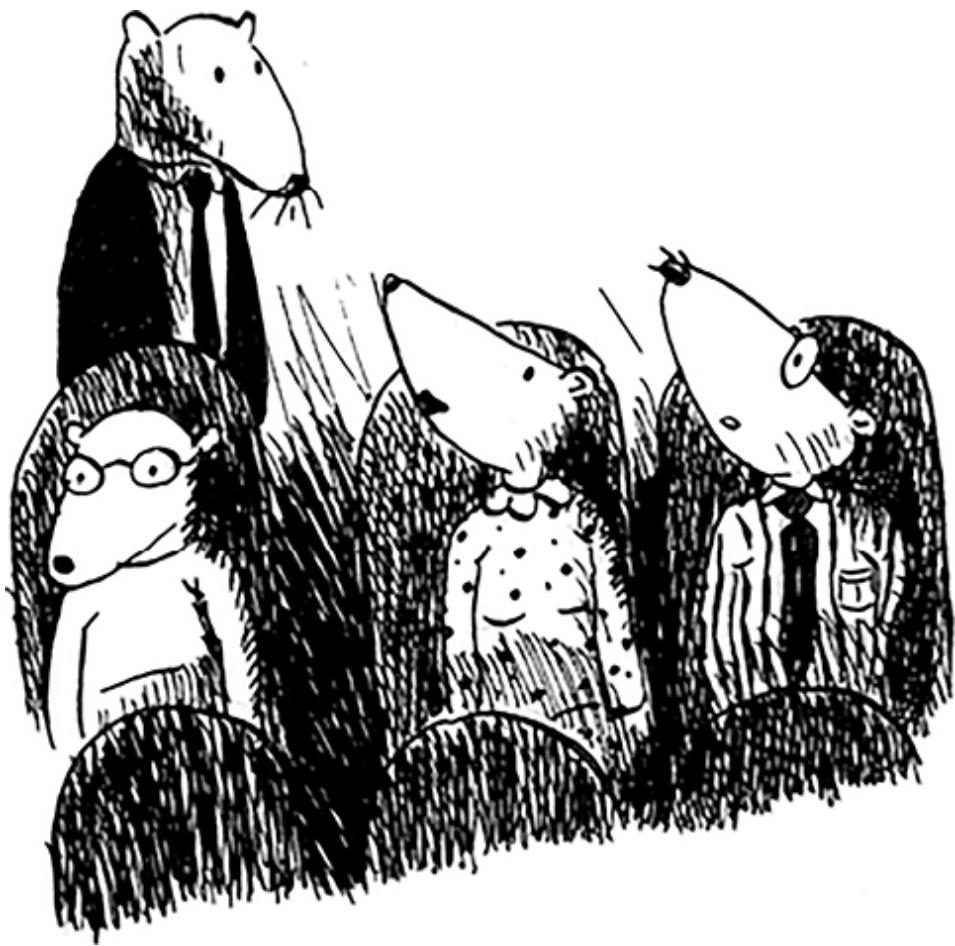
print(selectionSort([5, 3, 6, 2, 10]))
```

① Sorts an array

① Ordena una matriz

② Finds the smallest element in the array and adds it to the new array

② Encuentra el elemento más pequeño en la matriz y lo agrega a la nueva matriz



## **Recap**

### **Resumen**

- Your computer's memory is like a giant set of drawers.  
La memoria de su computadora es como un juego de cajones gigante.
- When you want to store multiple elements, use an array or a linked list.  
Cuando desee almacenar varios elementos, utilice una matriz o una lista vinculada.

- With an array, all your elements are stored right next to each other.

Con una matriz, todos sus elementos se almacenan uno al lado del otro.

- With a linked list, elements are strewn all over, and one element stores the address of the next one.

Con una lista enlazada, los elementos están esparcidos por todas partes y un elemento almacena la dirección del siguiente.

- Arrays allow fast reads.

Las matrices permiten lecturas rápidas.

- Linked lists allow fast inserts and deletes.

Las listas enlazadas permiten inserciones y eliminaciones rápidas.

# 3 Recursion

---

## 3 recursividad

---

### In this chapter

#### En este capítulo

- You learn about recursion. Recursion is a coding technique used in many algorithms. It's a building block for understanding later chapters in this book.

Aprendes sobre la recursividad. La recursividad es una técnica de codificación utilizada en muchos algoritmos. Es un elemento básico para comprender los capítulos posteriores de este libro.

- You learn what a base case and a recursive case is. The divide-and-conquer strategy (chapter 4) uses this simple concept to solve hard problems.

Aprende qué es un caso base y un caso recursivo. La estrategia divide y vencerás (capítulo 4) utiliza este concepto simple para resolver problemas difíciles.

I'm excited about this chapter because it covers *recursion*, an elegant way to solve problems. Recursion is one of my favorite topics, but it's divisive. People either love it or hate it—or hate it until they learn to love it a few years later. I personally was in that third camp. To make things easier for you, I have some advice:

Estoy entusiasmado con este capítulo porque cubre la recursividad, una forma elegante de resolver problemas. La recursividad es uno de mis temas favoritos, pero genera división. La gente lo ama o lo odia, o lo odia hasta que aprende a amarlo unos años más tarde. Yo personalmente estuve en ese tercer campo. Para facilitarte las cosas, tengo algunos consejos:

- This chapter has a lot of code examples. Run the code for yourself to see how it works.

Este capítulo tiene muchos ejemplos de código. Ejecute el código usted mismo para ver cómo funciona.

- I'll talk about recursive functions. At least once, step through a recursive function with pen and paper: something like, "Let's see, I pass 5 into `factorial`, and then I return five times passing 4 into `factorial`, which is . . . , and so on. Walking through a function like this will teach you how a recursive function works.

Hablaré de funciones recursivas. Al menos una vez, recorra una función recursiva con lápiz y papel: algo así como "Veamos, paso 5 a `factorial` y luego regreso cinco veces pasando 4 a `factorial`, que es . . . , etcétera.

Recorrer una función como esta le enseñará cómo funciona una función recursiva.

This chapter also includes a lot of pseudocode. *Pseudocode* is a high-level description in code of the problem you're trying to solve. It's written like code, but it's meant to be closer to human speech.

Este capítulo también incluye mucho pseudocódigo. El pseudocódigo es una descripción de alto nivel en código del problema que estás intentando resolver. Está escrito como código, pero pretende acercarse más al habla humana.

## ***Recursion***

### ***recursividad***

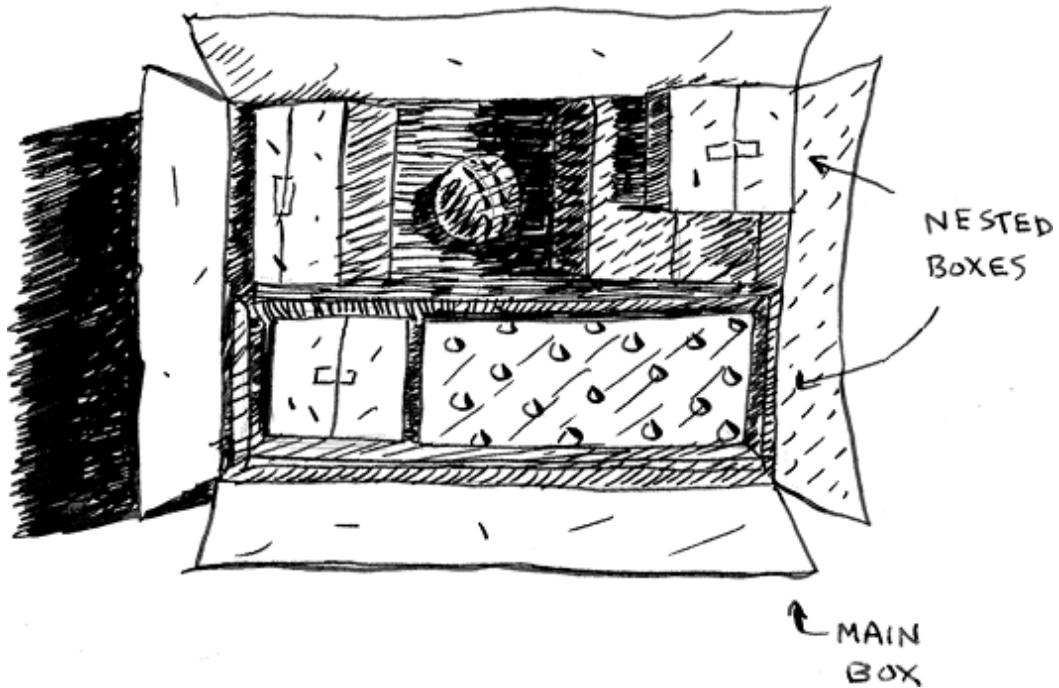
Suppose you're digging through your grandma's attic and come across a mysterious locked suitcase.

Supongamos que estás cavando en el ático de tu abuela y te encuentras con una misteriosa maleta cerrada con llave.



Grandma tells you that the key for the suitcase is probably in this other box.

La abuela te dice que probablemente la llave de la maleta esté en esta otra caja.

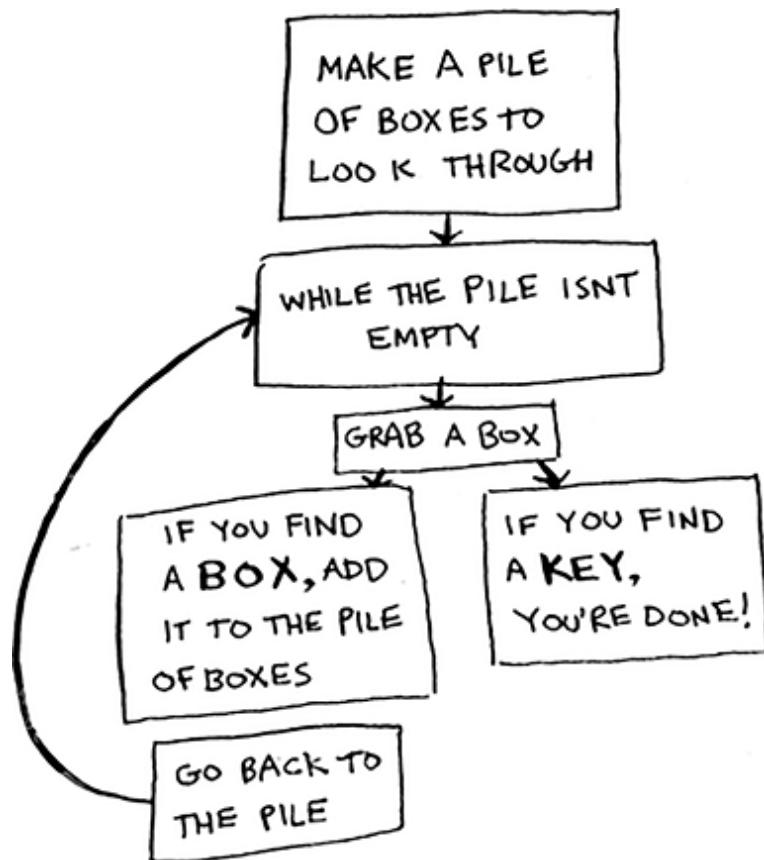


This box contains more boxes, with more boxes inside those boxes. The key is in a box somewhere. What's your algorithm to search for the key? Think of an algorithm before you read on.

Esta caja contiene más cajas, con más cajas dentro de esas cajas. La llave está en alguna caja en alguna parte. ¿Cuál es su algoritmo para buscar la clave? Piensa en un algoritmo antes de seguir leyendo.

Here's one approach:

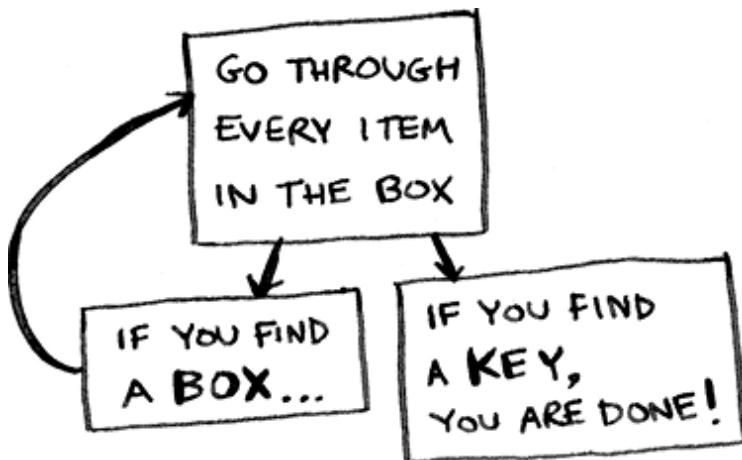
Aquí hay un enfoque:



1. Make a pile of boxes to look through.  
Haz un montón de cajas para mirar.
2. Grab a box and look through it.  
Coge una caja y mírala.
3. If you find a box, add it to the pile to look through later.  
Si encuentra una caja, agréguela a la pila para revisarla más tarde.
4. If you find a key, you're done!  
Si encuentras una llave, ¡ya está!
5. Repeat.  
Repetir.

Here's an alternate approach:

Aquí hay un enfoque alternativo:



1. Look through the box.

Mira a través de la caja.

2. If you find a box, go to step 1.

Si encuentra un cuadro, vaya al paso 1.

3. If you find a key, you're done!

Si encuentras una llave, ¡ya está!

Which approach seems easier to you? The first approach uses a `while` loop. While the pile isn't empty, grab a box and look through it. Here's some pseudocode:

¿Qué enfoque te parece más fácil? El primer enfoque utiliza un bucle `while`. Mientras la pila no esté vacía, toma una caja y mírala. Aquí hay un pseudocódigo:

```
def look_for_key(main_box):
    pile = main_box.make_a_pile_to_look_through()
    while pile is not empty:
        box = pile.grab_a_box()
        for item in box:
            if item.is_a_box():
                # Recurso: Go through every item in the box
                pass
            else:
                # Recurso: If you find a key, you're done!
                pass
```

```
pile.append(item)
elif item.is_a_key():
    print("found the key!")
```

The second way uses recursion. *Recursion* is where a function calls itself. Here's the second way in pseudocode:

La segunda forma utiliza la recursividad. La recursión es donde una función se llama a sí misma. Aquí está la segunda forma en pseudocódigo:

```
def look_for_key(box):
    for item in box:
        if item.is_a_box():
            look_for_key(item)      ①
        elif item.is_a_key():
            print("found the key!")
```

① Recursion!

① ¡Recursión!

Both approaches accomplish the same thing, but the second approach is clearer to me. Recursion is used when it makes the solution clearer. There's no performance benefit to using recursion; in fact, loops are sometimes better for performance. I like this quote by Leigh Caldwell on Stack Overflow: "Loops may achieve a performance gain for your program. Recursion may achieve a performance gain for your programmer. Choose which is more important in your situation!" (<http://stackoverflow.com/a/72694/139117>).

Ambos enfoques logran lo mismo, pero el segundo me resulta más claro. La recursividad se utiliza cuando aclara la solución. No hay ningún beneficio de rendimiento al usar la recursividad; de hecho, los bucles a veces son mejores para

el rendimiento. Me gusta esta cita de Leigh Caldwell sobre Stack Overflow: "Los bucles pueden lograr una ganancia de rendimiento para su programa. La recursividad puede lograr una ganancia de rendimiento para su programador. ¡Elige cuál es más importante en tu situación! (<http://stackoverflow.com/a/72694/139117>).

Many important algorithms use recursion, so it's important to understand the concept.

Muchos algoritmos importantes utilizan la recursividad, por lo que es importante comprender el concepto.

## ***Base case and recursive case***

### ***Caso base y caso recursivo***

Because a recursive function calls itself, it's easy to write a function incorrectly that ends up in an infinite loop.

Debido a que una función recursiva se llama a sí misma, es fácil escribir una función incorrectamente y terminar en un bucle infinito.



For example, suppose you want to write a function that prints a countdown like this:

Por ejemplo, supongamos que desea escribir una función que imprima una cuenta regresiva como esta:

```
> 3...2...1
```

You can write it recursively like so:

Puedes escribirlo recursivamente así:

```
def countdown(i):
    print(i)
    countdown(i-1)

countdown(3)
```

Write out this code and run it. You'll notice a problem: this function will run forever!

Escriba este código y ejecútelo. Notarás un problema: iesta función se ejecutará para siempre!



## Infinite loop

### Bucle infinito

```
> 3...2...1...0...-1...-2...
```

(Press Ctrl-C to kill your script.)

(Presione Ctrl-C para finalizar su secuencia de comandos).

When you write a recursive function, you have to tell it when to stop recursing. That's why *every recursive function has two parts: the base case and the recursive case*. The recursive case is when the function calls itself. The base case is when the function doesn't call itself again, so it doesn't go into an infinite loop.

Cuando escribes una función recursiva, debes decirle cuándo dejar de repetirse. Por eso toda función recursiva tiene dos partes: el caso base y el caso recursivo. El caso recursivo es cuando la función se llama a sí misma. El caso base es cuando la función no se vuelve a llamar a sí misma, por lo que no entra en un bucle infinito.

Let's add a base case to the countdown function:

Agreguemos un caso base a la función de cuenta regresiva:

```

def countdown(i):
    print(i)
    if i <= 1:          ①
        return
    else:              ②
        countdown(i-1)

countdown(3)

```

① Base case

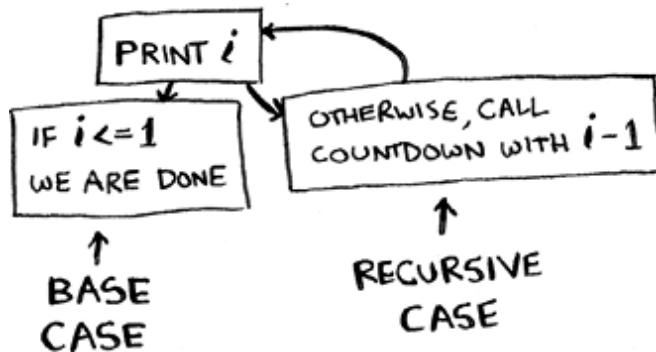
① Caso base

② Recursive case

② Caso recursivo

Now the function works as expected. It goes something like this.

Ahora la función funciona como se esperaba. Es algo parecido a esto.



## The stack

## La pila

This section covers the *call stack*. The call stack is an important concept in general programming, and it's also

important to understand when using recursion.

Esta sección cubre la pila de llamadas. La pila de llamadas es un concepto importante en la programación general y también es importante comprenderlo cuando se utiliza la recursividad.



Suppose you're throwing a barbecue. You keep a to-do list for the barbecue, in the form of a stack of sticky notes.

Supongamos que estás haciendo una barbacoa. Mantienes una lista de cosas por hacer para la barbacoa, en forma de una pila de notas adhesivas.



Remember back when we talked about arrays and lists, and you had a to-do list? You could add to-do items anywhere to the list or delete random items. The stack of sticky notes is much simpler. When you insert an item, it gets added to the

top of the list. When you read an item, you only read the topmost item, and it's taken off the list. So your to-do list has only two actions: *push* (insert) and *pop* (remove and read).

¿Recuerdas cuando hablábamos de matrices y listas y tenías una lista de tareas pendientes? Puede agregar tareas pendientes en cualquier lugar de la lista o eliminar elementos aleatorios. La pila de notas adhesivas es mucho más sencilla. Cuando inserta un elemento, se agrega al principio de la lista. Cuando lees un elemento, solo lees el elemento superior y se elimina de la lista. Entonces, su lista de tareas pendientes tiene solo dos acciones: presionar (insertar) y hacer estallar (eliminar y leer).



**PUSH**  
(ADD A NEW ITEM  
TO THE TOP)



**POP**  
(REMOVE THE TOPMOST  
ITEM AND READ IT)

Let's see the to-do list in action.

Veamos la lista de tareas pendientes en acción.



This data structure is called a *stack*. The stack is a simple data structure. You've been using a stack this whole time without realizing it!

Esta estructura de datos se llama pila. La pila es una estructura de datos simple. ¡Has estado usando una pila todo este tiempo sin darte cuenta!

## The call stack

### La pila de llamadas

Your computer uses a stack internally called the *call stack*. Let's see it in action. Here's a simple function:

Su computadora utiliza una pila internamente llamada pila de llamadas. Veámoslo en acción. Aquí hay una función simple:

```
def greet(name):
    print("hello, " + name + "!")
    greet2(name)
```

```
print("getting ready to say bye...")
bye()
```

This function greets you and then calls two other functions.  
Here are those two functions:

Esta función lo saluda y luego llama a otras dos funciones.  
Aquí están esas dos funciones:

```
def greet2(name):
    print("how are you, " + name + "?")

def bye():
    print("ok bye! ")
```

Let's walk through what happens when you call a function.

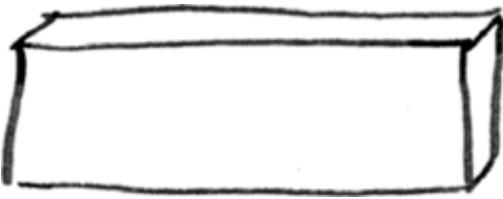
Veamos lo que sucede cuando llamas a una función.

**NOTE** To keep things simple, I'm only showing the calls to `greet`, `greet2`, and `bye`. I'm not showing the calls to the `print` function.

Nota Para simplificar las cosas, solo muestro las llamadas a `greet`, `greet2` y `bye`. No muestro las llamadas a la función `print`.

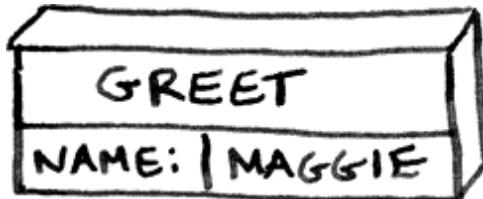
Suppose you call `greet("maggie")`. First, your computer allocates a box of memory for that function call.

Supongamos que llamas a `greet("maggie")`. Primero, su computadora asigna una caja de memoria para esa llamada de función.



Now let's use the memory. The variable `name` is set to "maggie". That needs to be saved in memory.

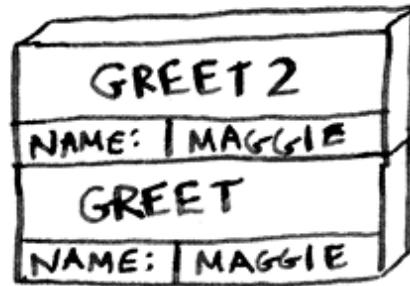
Ahora usemos la memoria. La variable `name` se establece en "maggie". Eso debe guardarse en la memoria.



Every time you make a function call, your computer saves the values for all the variables for that call in memory like this. Next, you print `hello, maggie!` Then you call `greet2("maggie")`. Again, your computer allocates a box of memory for this function call.

Cada vez que realiza una llamada a una función, su computadora guarda los valores de todas las variables para esa llamada en la memoria de esta manera. A continuación, imprime `hello, maggie!` Luego llama a `greet2("maggie")`. Nuevamente, su computadora asigna un cuadro de memoria para esta llamada de función.

CURRENT  
FUNCTION  
CALL



Your computer is using a stack for these boxes. The second box is added on top of the first one. You print how are you, maggie? Then you return from the function call. When this happens, the box on top of the stack gets popped off.

Su computadora está usando una pila para estas cajas. El segundo cuadro se agrega encima del primero. Imprimes how are you, maggie? Luego regresas de la llamada a la función. Cuando esto sucede, la caja en la parte superior de la pila se desprende.



Now the topmost box on the stack is for the `greet` function, which means you returned to the `greet` function. When you called the `greet2` function, the `greet` function was *partially completed*. This is the big idea behind this section: *when you call a function from another function, the calling function is*

*paused in a partially completed state.* All the values of the variables for that function are still stored on the call stack (i.e., in memory). Now that you're done with the `greet2` function, you're back to the `greet` function, and you pick up where you left off. First, you print `getting ready to say bye...` Then you call the `bye` function.

Ahora el cuadro superior de la pila es para la función `greet`, lo que significa que regresó a la función `greet`. Cuando llamó a la función `greet2`, la función `greet` se completó parcialmente. Esta es la gran idea detrás de esta sección: cuando llamas a una función desde otra función, la función que llama se pausa en un estado parcialmente completado. Todos los valores de las variables para esa función todavía están almacenados en la pila de llamadas (es decir, en la memoria). Ahora que ha terminado con la función `greet2`, regresa a la función `greet` y continúa donde lo dejó. Primero, imprime `getting ready to say bye...` Luego llama a la función `bye`.



A box for that function is added to the top of the stack. Then you print `ok bye!` and return from the function call.

Se agrega un cuadro para esa función en la parte superior de la pila. Luego imprime `ok bye!` y regresa de la llamada a la función.



And you're back to the `greet` function. There's nothing else to be done, so you return from the `greet` function, too. This stack, used to save the variables for multiple functions, is called the *call stack*.

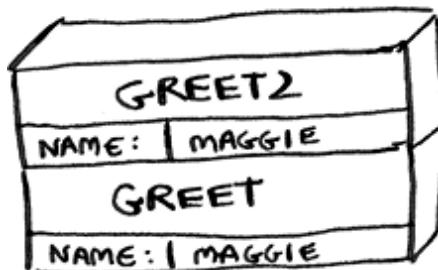
Y volverás a la función `greet`. No hay nada más que hacer, por lo que también regresa desde la función `greet`. Esta pila, utilizada para guardar las variables para múltiples funciones, se llama pila de llamadas.

## EXERCISE

### EJERCICIO

**3.1** Suppose I show you a call stack like this.

3.1 Supongamos que les muestro una pila de llamadas como esta.



What information can you give me, just based on this call stack?

¿Qué información puede darme, basándose únicamente en esta pila de llamadas?

Now let's see the call stack in action with a recursive function.

Ahora veamos la pila de llamadas en acción con una función recursiva.

## The call stack with recursion

### La pila de llamadas con recursividad

Recursive functions use the call stack, too! Let's look at this in action with the `factorial` function. `factorial(5)` is written as  $5!$ , and it's defined like this:  $5! = 5 * 4 * 3 * 2 * 1$ . Similarly, `factorial(3)` is  $3 * 2 * 1$ . Here's a recursive function to calculate the factorial of a number:

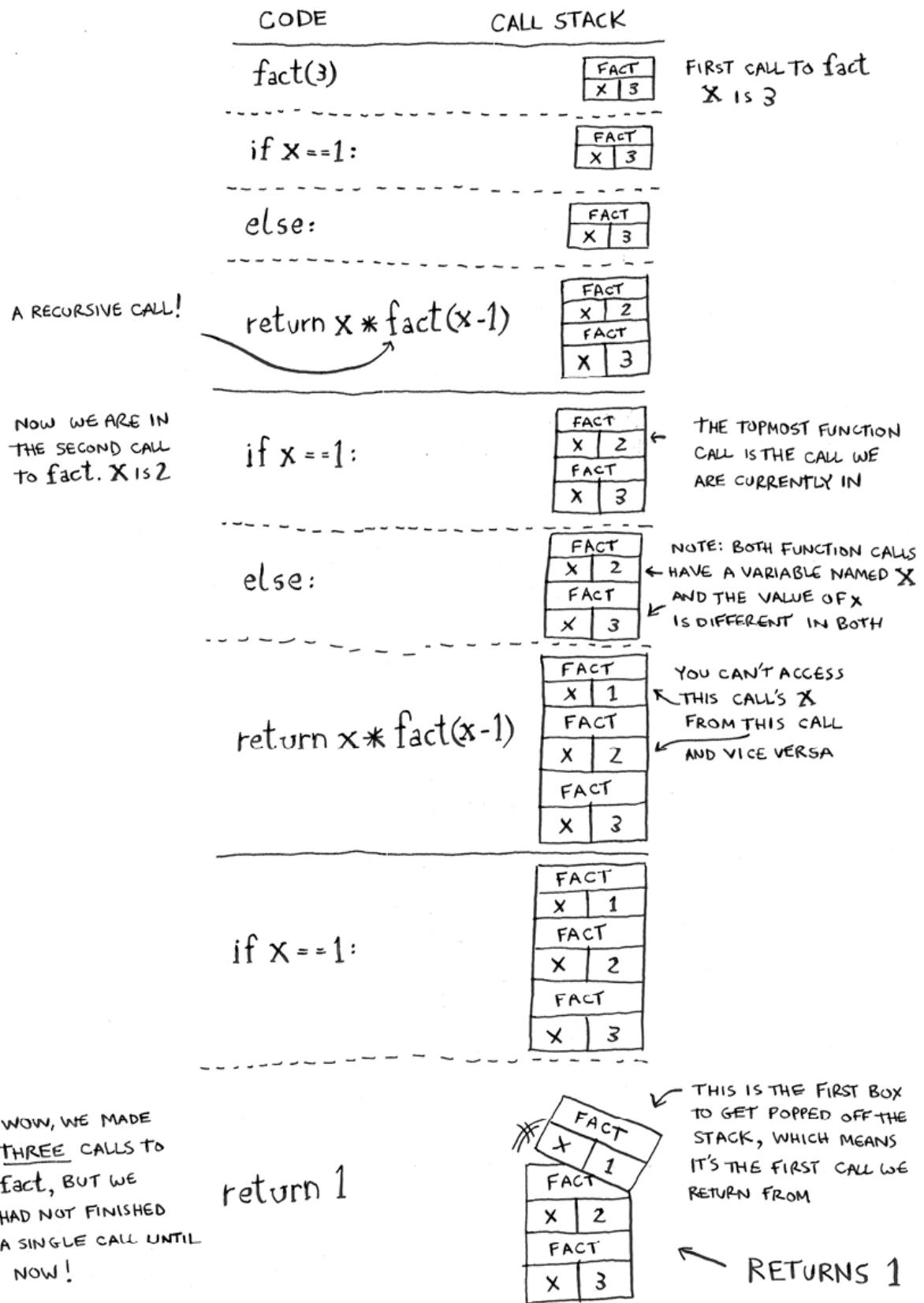
Las funciones recursivas también usan la pila de llamadas! Veamos esto en acción con la función `factorial`.

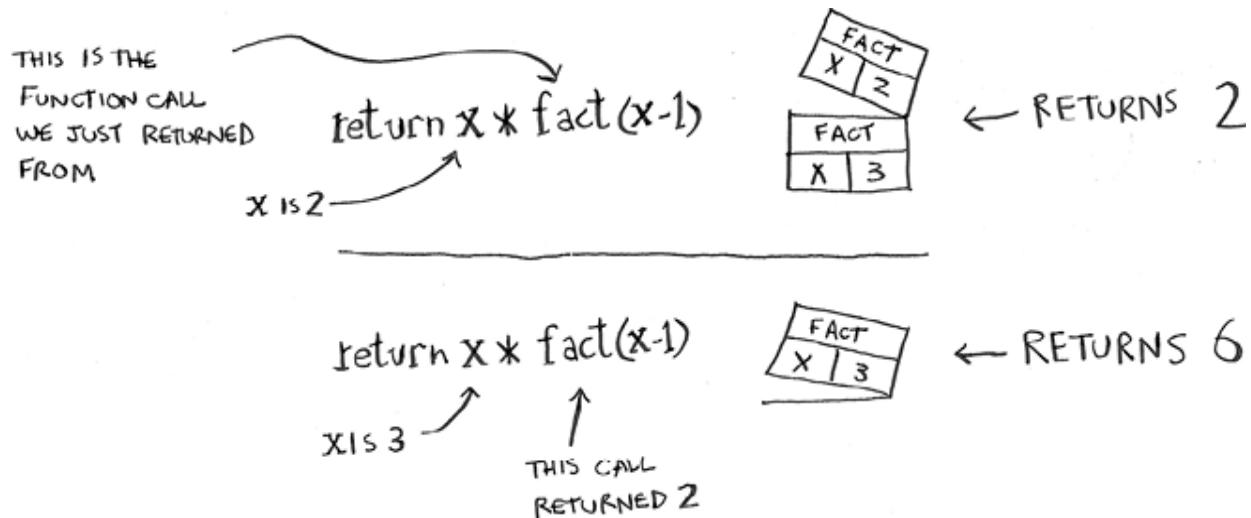
`factorial(5)` se escribe como  $5!$ , y se define así:  $i5! = 5 * 4 * 3 * 2 * 1$ . De manera similar, `factorial(3)` es  $3 * 2 * 1$ . Aquí hay una función recursiva para calcular el factorial de un número:

```
def fact(x):
    if x == 1:
        return 1
    else:
        return x * fact(x-1)
```

Now you can call `fact(3)`. Let's step through this call line by line and see how the stack changes. Remember, the topmost box in the stack tells you what call to `fact` you're currently on.

Ahora puedes llamar a `fact(3)`. Repasemos esta llamada línea por línea y veamos cómo cambia la pila. Recuerde, el cuadro superior de la pila le indica en qué llamada a `fact` se encuentra actualmente.



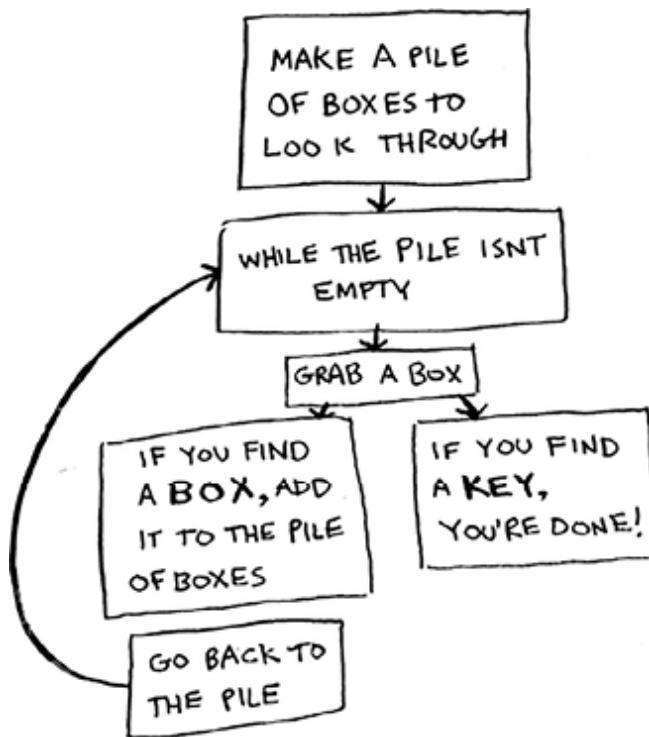


Notice that each call to `fact` has its own copy of `x`. You can't access a different function's copy of `x`.

Observe que cada llamada a `fact` tiene su propia copia de `x`. No puedes acceder a la copia de una función diferente de `x`.

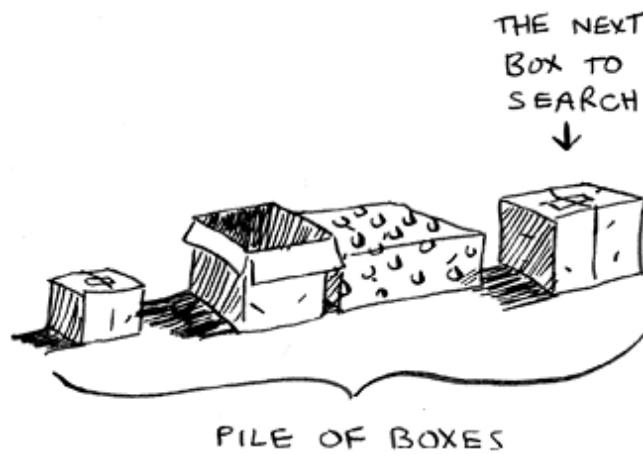
The stack plays a big part in recursion. In the opening example, there were two approaches to finding the key. Here's the first way again.

La pila juega un papel importante en la recursividad. En el ejemplo inicial, había dos enfoques para encontrar la clave. Aquí está la primera forma nuevamente.



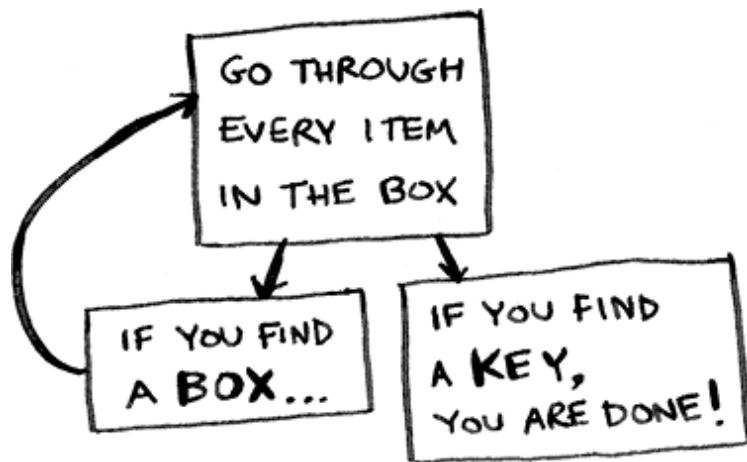
This way, you make a pile of boxes to search through, so you always know what boxes you still need to search.

De esta manera, crea un montón de cuadros para buscar, de modo que siempre sepa qué cuadros aún necesita buscar.



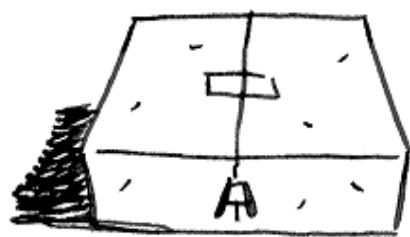
But in the recursive approach, there's no pile.

Pero en el enfoque recursivo no hay montón.

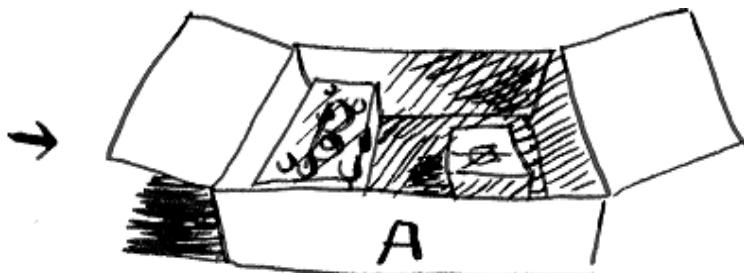


If there's no pile, how does your algorithm know what boxes you still have to look through? Here's an example.

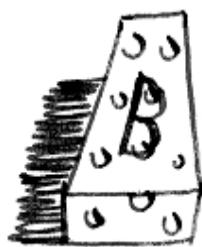
Si no hay ninguna pila, ¿cómo sabe su algoritmo qué casillas le quedan por revisar? He aquí un ejemplo.



YOU LOOK THROUGH  
BOX A



INSIDE YOU FIND  
BOXES B AND C



YOU CHECK  
BOX B



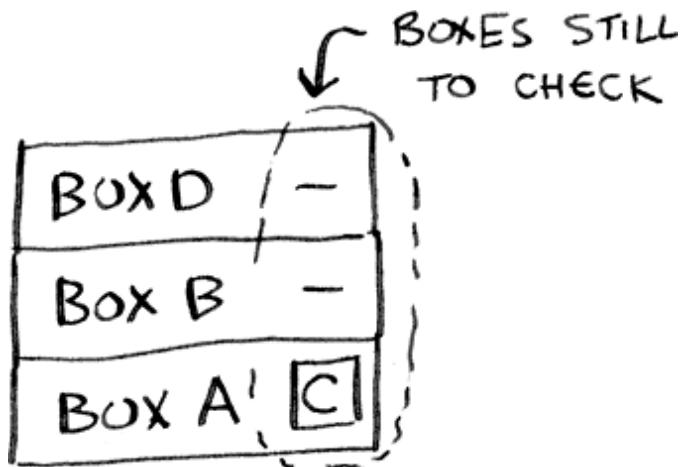
IT CONTAINS  
BOX D



IT IS  
EMPTY

At this point, the call stack looks like this.

En este punto, la pila de llamadas tiene este aspecto.



The “pile of boxes” is saved on the stack! This is a stack of half-completed function calls, each with its own half-complete list of boxes to look through. Using the stack is convenient because you don’t have to keep track of a pile of boxes yourself—the stack does it for you.

¡El “montón de cajas” se guarda en la pila! Esta es una pila de llamadas a funciones a medio completar, cada una con su propia lista de cuadros a medio completar para revisar. Usar la pila es conveniente porque usted no tiene que realizar un seguimiento de una pila de cajas: la pila lo hace por usted.

Using the stack is convenient, but there’s a cost: saving all that info can take up a lot of memory. Each of those function calls takes up some memory, and when your stack is too tall, that means your computer is saving information for many function calls. At that point, you have two options:

Usar la pila es conveniente, pero tiene un costo: guardar toda esa información puede consumir mucha memoria. Cada una de esas llamadas a funciones ocupa algo de memoria y, cuando su pila es demasiado alta, eso significa que su

computadora está guardando información para muchas llamadas a funciones. En ese punto, tienes dos opciones:

- You can rewrite your code to use a loop instead.  
Puedes reescribir tu código para usar un bucle en su lugar.
- You can use something called *tail recursion*. That's an advanced recursion topic that is out of the scope of this book. It's also only supported by some languages, not all.

Puedes usar algo llamado recursividad de cola. Éste es un tema de recursión avanzada que está fuera del alcance de este libro. Además, sólo es compatible con algunos idiomas, no con todos.

## EXERCISE

### EJERCICIO

**3.2** Suppose you accidentally write a recursive function that runs forever. As you saw, your computer allocates memory on the stack for each function call. What happens to the stack when your recursive function runs forever?

3.2 Suponga que accidentalmente escribe una función recursiva que se ejecuta para siempre. Como vio, su computadora asigna memoria en la pila para cada llamada de función. ¿Qué sucede con la pila cuando su función recursiva se ejecuta para siempre?

## **Recap**

### **Resumen**

- Recursion is when a function calls itself.  
La recursividad es cuando una función se llama a sí misma.
- Every recursive function has two cases: the base case and the recursive case.  
Toda función recursiva tiene dos casos: el caso base y el caso recursivo.
- A stack has two operations: push and pop.  
Una pila tiene dos operaciones: empujar y hacer estallar.
- All function calls go onto the call stack.  
Todas las llamadas a funciones van a la pila de llamadas.
- The call stack can get very large, which takes up a lot of memory.  
La pila de llamadas puede llegar a ser muy grande, lo que ocupa mucha memoria.



# 4 Quicksort

---

## 4 clasificación rápida

---

### In this chapter

#### En este capítulo

- You learn about divide and conquer. Sometimes you'll come across a problem that can't be solved by any algorithm you've learned. When a good algorithmist encounters such a problem, they don't just give up. They have a toolbox full of techniques they use on the problem, trying to come up with a solution. Divide and conquer is the first general technique you learn.  
Aprendes sobre divide y vencerás. A veces te encontrarás con un problema que no puede resolverse con ningún algoritmo que hayas aprendido. Cuando un buen algorítmico se encuentra con un problema de este tipo, no se da por vencido. Tienen una caja de herramientas llena de técnicas que utilizan para resolver el problema, tratando de encontrar una solución. Divide y vencerás es la primera técnica general que aprendes.
- You learn about quicksort, an elegant sorting algorithm often used in practice. Quicksort uses divide and conquer.  
Aprenderá sobre Quicksort, un elegante algoritmo de clasificación que se utiliza a menudo en la práctica. Quicksort utiliza divide y vencerás.

You learned all about recursion in the last chapter. This chapter focuses on using your new skill to solve problems.

We'll explore *divide and conquer* (D&C), a well-known recursive technique for solving problems.

Aprendiste todo sobre la recursividad en el último capítulo. Este capítulo se centra en el uso de su nueva habilidad para resolver problemas. Exploraremos divide y vencerás (D&C), una técnica recursiva muy conocida para resolver problemas.

This chapter really gets into the meat of algorithms. After all, an algorithm isn't very useful if it can only solve one type of problem. Instead, D&C gives you a new way to think about solving problems. D&C is another tool in your toolbox. When you get a new problem, you don't have to be stumped. Instead, you can ask, "Can I solve this if I use divide and conquer?"

Este capítulo realmente profundiza en los algoritmos. Después de todo, un algoritmo no es muy útil si sólo puede resolver un tipo de problema. En cambio, D&C le ofrece una nueva forma de pensar en la resolución de problemas. D&C es otra herramienta en su caja de herramientas. Cuando tienes un nuevo problema, no tienes por qué quedarte perplejo. En su lugar, puedes preguntar: "¿Puedo resolver esto si uso divide y vencerás?".

At the end of the chapter, you'll learn your first major D&C algorithm: *quicksort*. Quicksort is a sorting algorithm that is much faster than selection sort (which you learned in chapter 2). It's a good example of elegant code.

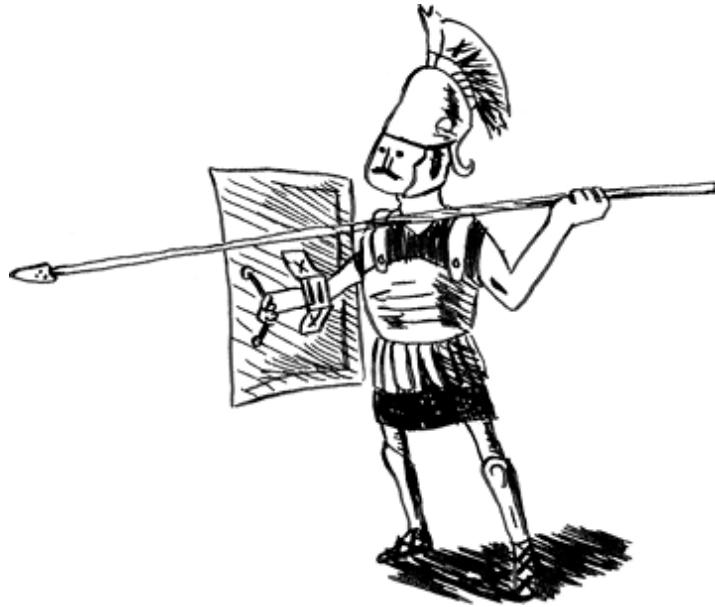
Al final del capítulo, aprenderá su primer algoritmo importante de D&C: clasificación rápida. Quicksort es un algoritmo de clasificación mucho más rápido que la clasificación por selección (que aprendió en el capítulo 2). Es un buen ejemplo de código elegante.

## ***Divide and conquer***

## ***Divide y conquistarás***

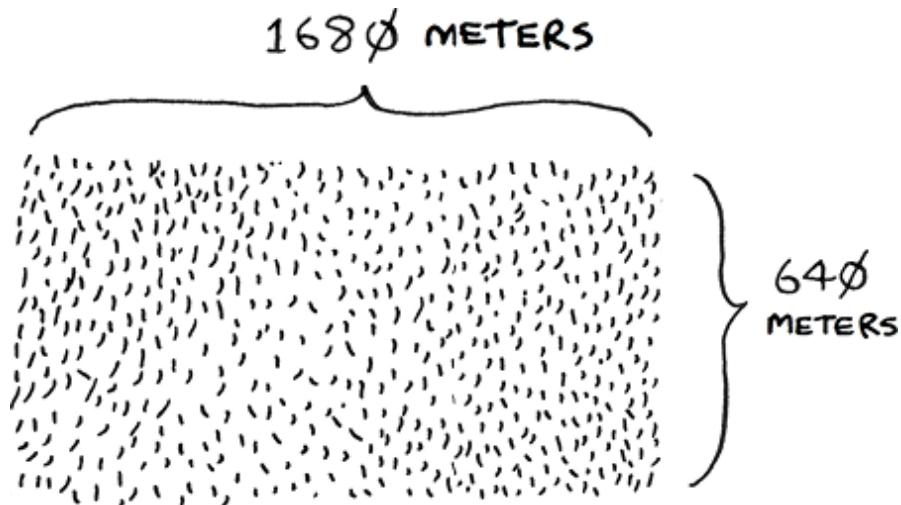
D&C can take some time to grasp. So, we'll do three examples. First, I'll show you a visual example. Then I'll show a code example that is less pretty but maybe easier. Finally, we'll go through quicksort, a sorting algorithm that uses D&C.

D&C puede tardar algún tiempo en comprenderse. Entonces, haremos tres ejemplos. Primero, les mostraré un ejemplo visual. Luego mostraré un ejemplo de código que es menos bonito pero quizás más fácil. Finalmente, veremos la clasificación rápida, un algoritmo de clasificación que utiliza D&C.



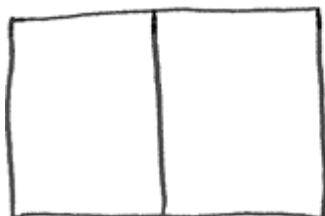
Suppose you're a farmer with a plot of land.

Supongamos que es un agricultor con un terreno.

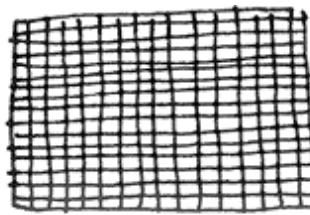


You want to divide this farm evenly into *square* plots. You want the plots to be as big as possible. So none of these will work.

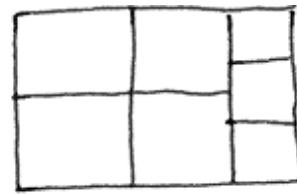
Quiere dividir esta granja en partes iguales en parcelas cuadradas. Quieres que las tramas sean lo más grandes posible. Entonces ninguno de estos funcionará.



BOXES ARE  
NOT SQUARE



BOXES ARE TOO  
SMALL



ALL BOXES MUST  
BE SAME SIZE

How do you figure out the largest square size you can use for a plot of land? Use the D&C strategy! D&C algorithms are recursive algorithms. There are two steps to solving a problem using D&C:

¿Cómo se calcula el tamaño cuadrado más grande que se puede utilizar para un terreno? ¡Usa la estrategia D&C! Los algoritmos D&C son algoritmos recursivos. Hay dos pasos para resolver un problema usando D&C:

1. Figure out the base case. This should be the simplest possible case.

Descubra el caso base. Este debería ser el caso más sencillo posible.

2. Divide or decrease your problem until it becomes the base case.

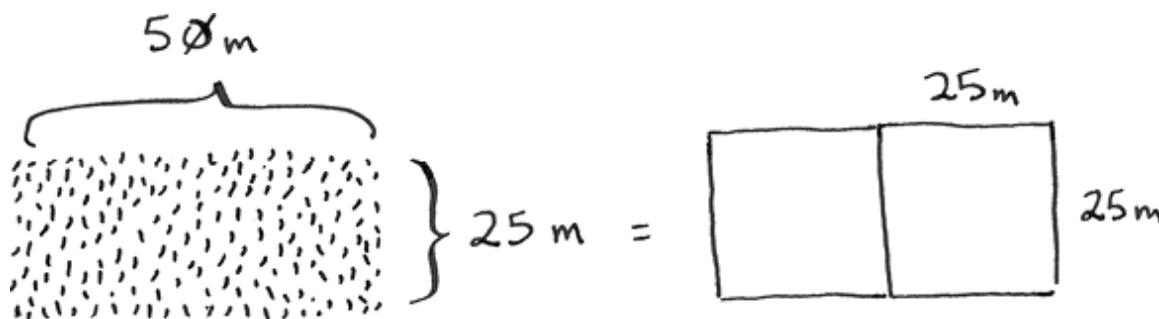
Divida o disminuya su problema hasta que se convierta en el caso base.

Let's use D&C to find the solution to this problem. What is the largest square size you can use?

Usemos D&C para encontrar la solución a este problema. ¿Cuál es el tamaño de cuadrado más grande que puedes usar?

First, figure out the base case. The easiest case would be if one side was a multiple of the other side.

Primero, determine el caso base. El caso más sencillo sería si un lado fuera múltiplo del otro lado.



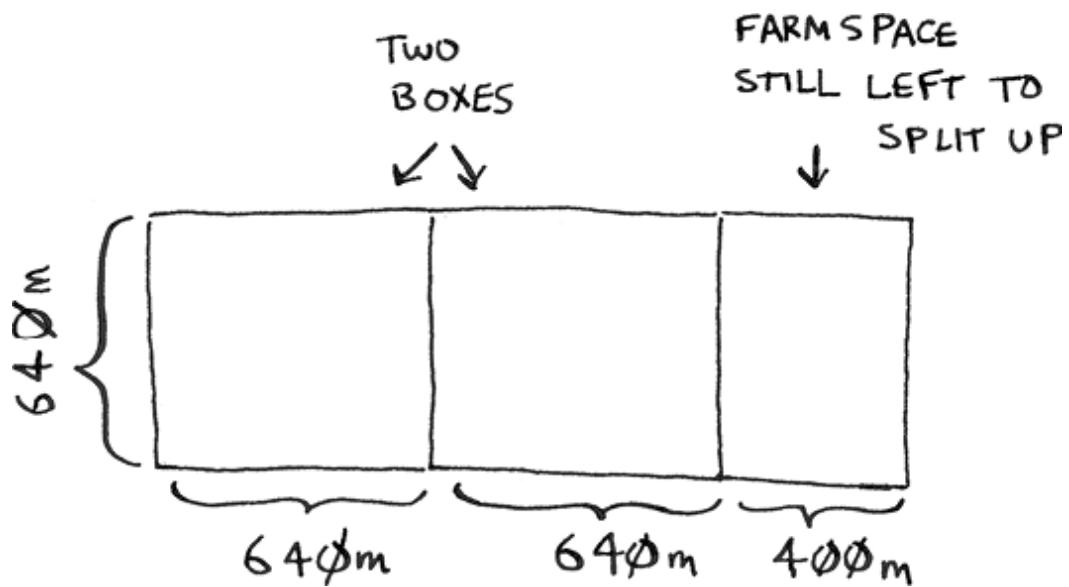
Suppose one side is 25 m and the other side is 50 m. Then the largest box you can use is 25 m  $\times$  25 m. You need two of those boxes to divide up the land.

Supongamos que un lado mide 25 m y el otro lado mide 50 m. Entonces la caja más grande que puedes usar es de 25 m  $\times$  25 m. Necesitas dos de esas cajas para dividir el terreno.

Now you need to figure out the recursive case. This is where D&C comes in. According to D&C, with every recursive call, you have to reduce your problem. How do you reduce the

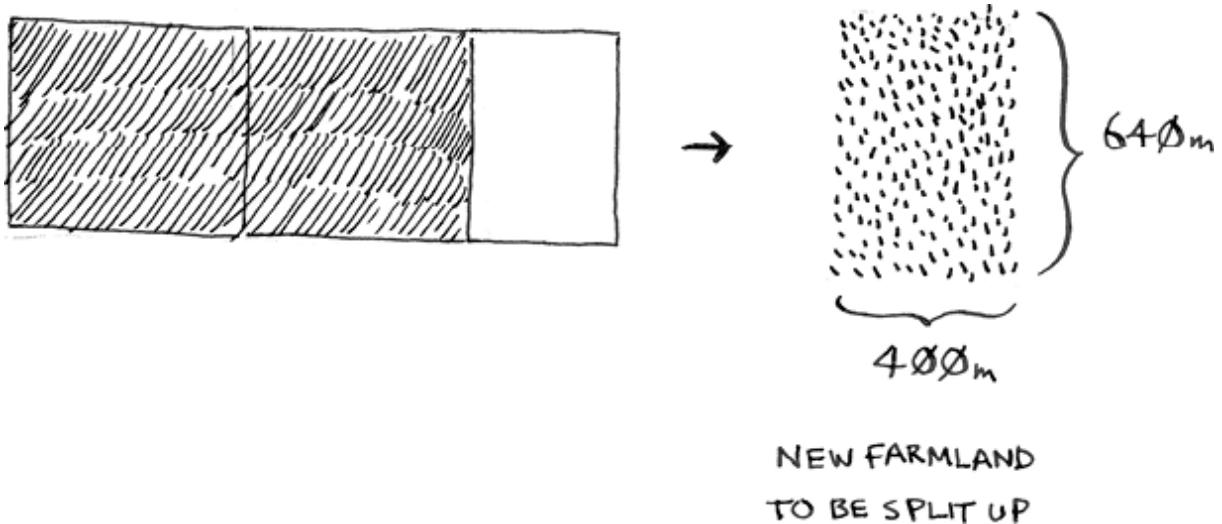
problem here? Let's start by marking out the biggest boxes you can use.

Ahora necesitas descubrir el caso recursivo. Aquí es donde entra en juego D&C. Según D&C, con cada llamada recursiva, debes reducir el problema. ¿Cómo se reduce el problema aquí? Comencemos marcando las cajas más grandes que puedes usar.



You can fit two  $640 \text{ m} \times 640 \text{ m}$  boxes in there, and there's some land still left to be divided. Now here comes the "Aha!" moment. There's a farm segment left to divide. *Why don't you apply the same algorithm to this segment?*

Allí caben dos cajas de  $640 \text{ m} \times 640 \text{ m}$  y todavía queda algo de terreno por dividir. Ahora aquí viene el "¡Ajá!" momento. Queda un segmento agrícola por dividir. ¿Por qué no aplica el mismo algoritmo a este segmento?



So you started out with a  $1,680 \text{ m} \times 640 \text{ m}$  farm that needed to be split up. But now you need to split up a smaller segment,  $640 \text{ m} \times 400 \text{ m}$ . If you *find the biggest box that will work for this size, that will be the biggest box that will work for the entire farm*. You just reduced the problem from a  $1,680 \text{ m} \times 640 \text{ m}$  farm to a  $640 \text{ m} \times 400 \text{ m}$  farm!

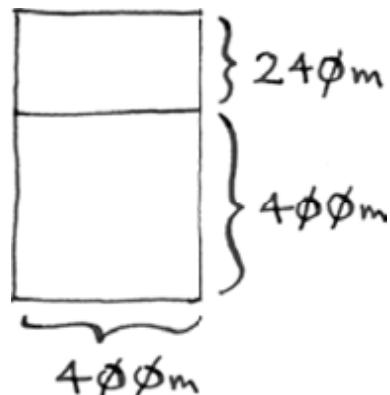
Así que empezaste con una finca de  $1,680 \text{ m} \times 640 \text{ m}$  que necesitaba ser dividida. Pero ahora necesitas dividir un segmento más pequeño,  $640 \text{ m} \times 400 \text{ m}$ . Si encuentra la caja más grande que funcionará para este tamaño, esa será la caja más grande que funcionará para toda la granja. ¡Acaba de reducir el problema de una granja de  $1,680 \text{ m} \times 640 \text{ m}$  a una granja de  $640 \text{ m} \times 400 \text{ m}$ !

### Euclid's algorithm

### El algoritmo de Euclides

"If you find the biggest box that will work for this size, that will be the biggest box that will work for the entire farm." If it's not obvious to you why this statement is true, don't worry. It isn't obvious. Unfortunately, the proof for why it works is a little too long to include in this book, so you'll just have to believe me that it works. If you want to understand the proof, look up Euclid's algorithm for finding the greatest common denominator. The Khan Academy has a good explanation (<http://mng.bz/orm2>).

"Si encuentra la caja más grande que funcione para este tamaño, será la caja más grande que funcionará para toda la granja". Si no le resulta obvio por qué esta afirmación es cierta, no se preocupe. No es obvio. Desafortunadamente, la prueba de por qué funciona es demasiado larga para incluirla en este libro, así que tendrás que creerme que funciona. Si quieres entender la prueba, busca el algoritmo de Euclides para encontrar el máximo común denominador. La Khan Academy tiene una buena explicación (<http://mng.bz/orm2>).

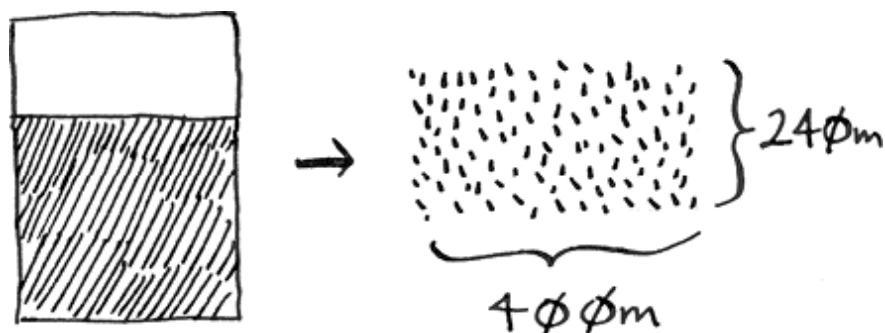


Let's apply the same algorithm again. Starting with a  $640\text{ m} \times 400\text{ m}$  farm, the biggest box you can create is  $400\text{ m} \times 400\text{ m}$ .

Apliquemos el mismo algoritmo nuevamente. Comenzando con una finca de  $640\text{ m} \times 400\text{ m}$ , la caja más grande que puede crear es de  $400\text{ m} \times 400\text{ m}$ .

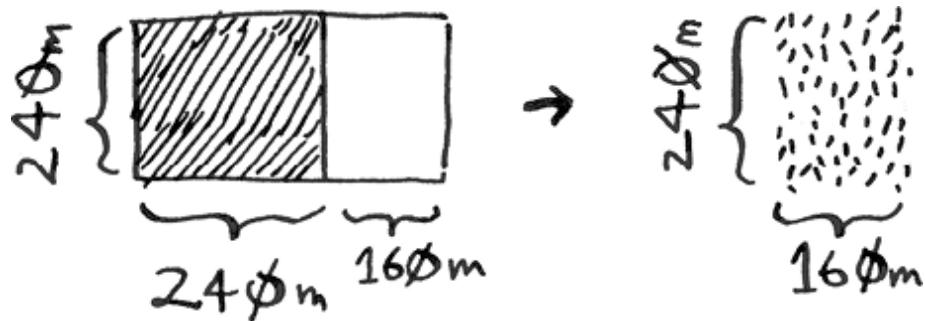
And that leaves you with a smaller segment,  $400\text{ m} \times 240\text{ m}$ .

Y eso te deja con un segmento más pequeño,  $400 \text{ m} \times 240 \text{ m}$ .



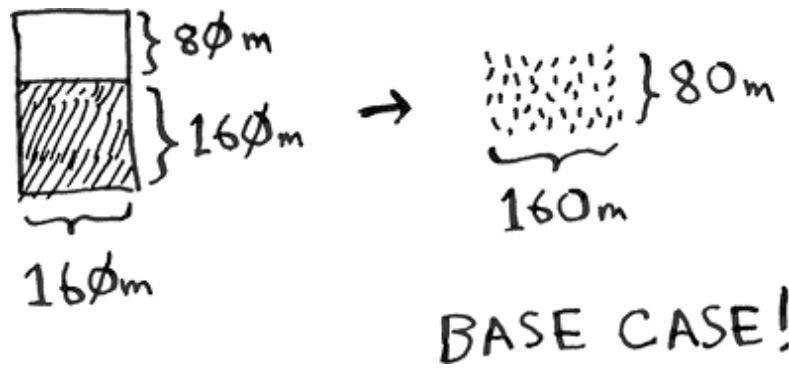
And you can draw a box on that to get an even smaller segment,  $240 \text{ m} \times 160 \text{ m}$ .

Y puedes dibujar un cuadro sobre eso para obtener un segmento aún más pequeño,  $240 \text{ m} \times 160 \text{ m}$ .



And then you draw a box on that to get an even *smaller* segment.

Y luego dibujas un cuadro sobre eso para obtener un segmento aún más pequeño.



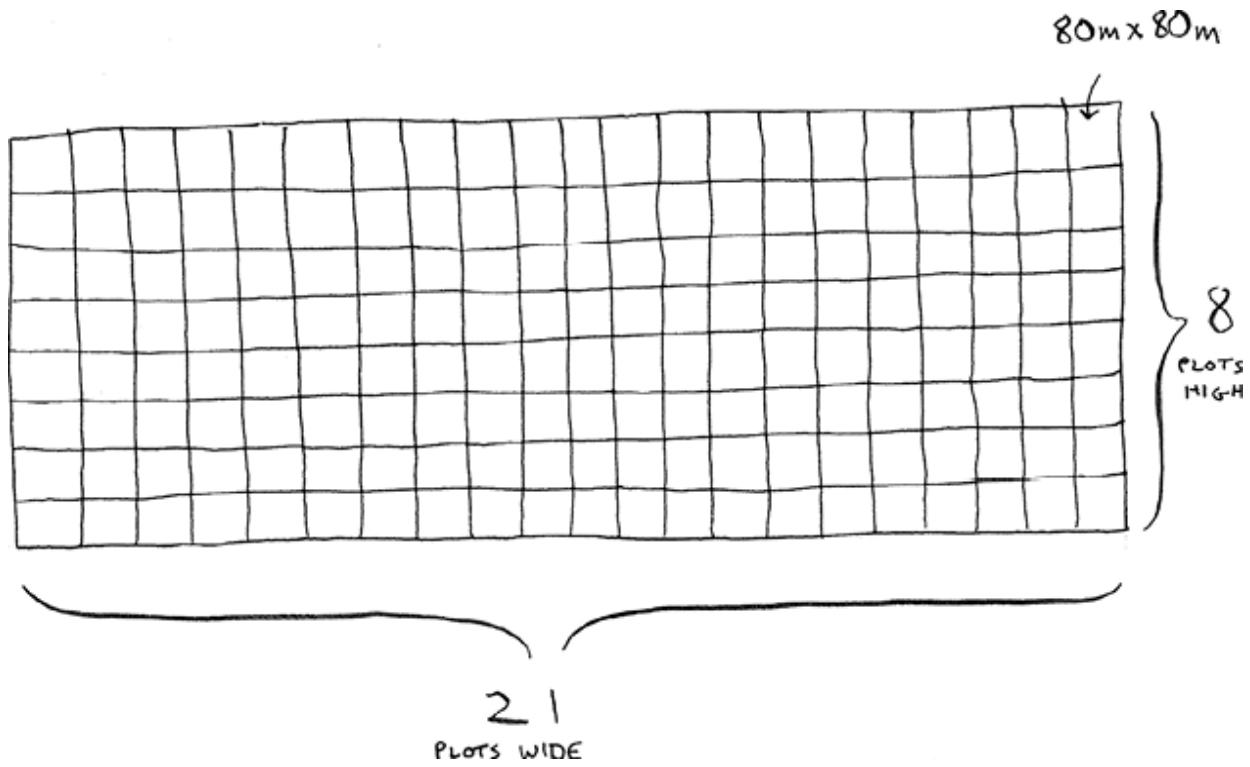
Hey, you're at the base case: 80 is a factor of 160. If you split up this segment using boxes, you don't have anything left over!

Oye, estás en el caso base: 80 es un factor de 160. Si divides este segmento usando cuadros, no te sobrará nada!



So, for the original farm, the biggest plot size you can use is  $80 \text{ m} \times 80 \text{ m}$ .

Entonces, para la finca original, el tamaño de parcela más grande que puede usar es  $80 \text{ m} \times 80 \text{ m}$ .



To recap, here's how D&C works:

En resumen, así es como funciona D&C:

1. Figure out a simple case as the base case.  
Encuentre un caso simple como caso base.
2. Figure out how to reduce your problem and get to the base case.  
Descubra cómo reducir su problema y llegar al caso base.

D&C isn't a simple algorithm that you can apply to a problem. Instead, it's a way to think about a problem. Let's do one more example.

D&C no es un algoritmo simple que puedas aplicar a un problema. Más bien, es una forma de pensar en un

problema. Hagamos un ejemplo más.



You're given an array of numbers. You have to add up all the numbers and return the total. It's pretty easy to do this with a loop:

Te dan una serie de números. Tienes que sumar todos los números y devolver el total. Es bastante fácil hacer esto con un bucle:

```
def sum(arr):
    total = 0
    for x in arr:
        total += x
    return total

print(sum([1, 2, 3, 4]))
```

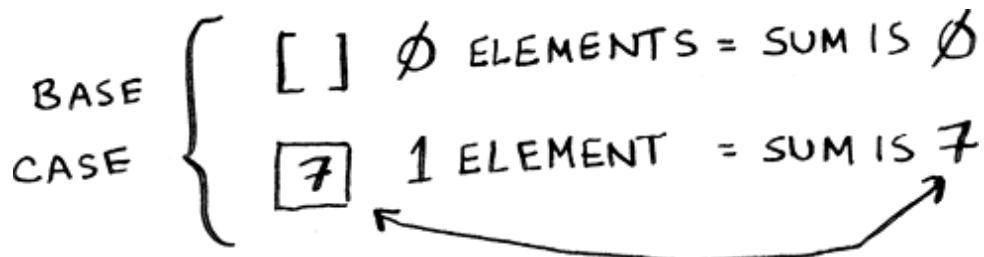
But how would you do this with a recursive function?

Pero, ¿cómo harías esto con una función recursiva?

**Step 1:** Figure out the base case. What's the simplest array you could get? Think about the simplest case, and then read on. If you get an array with 0 or 1 element, that's pretty easy to sum up.

Paso 1: Descubra el caso base. ¿Cuál es la matriz más simple que puedes conseguir?  
Piensa en el caso más sencillo y luego sigue

leyendo. Si obtiene una matriz con 0 o 1 elemento, es bastante fácil de resumir.



So that will be the base case.

Entonces ese será el caso base.

**Step 2:** You need to move closer to an empty array with every recursive call. How do you reduce your problem size? Here's one way.

Paso 2: Debes acercarte a una matriz vacía con cada llamada recursiva. ¿Cómo reduce el tamaño de su problema? Esta es una forma.

$$\text{sum}([2|4|6]) = 12$$

It's the same as this.

Es lo mismo que esto.

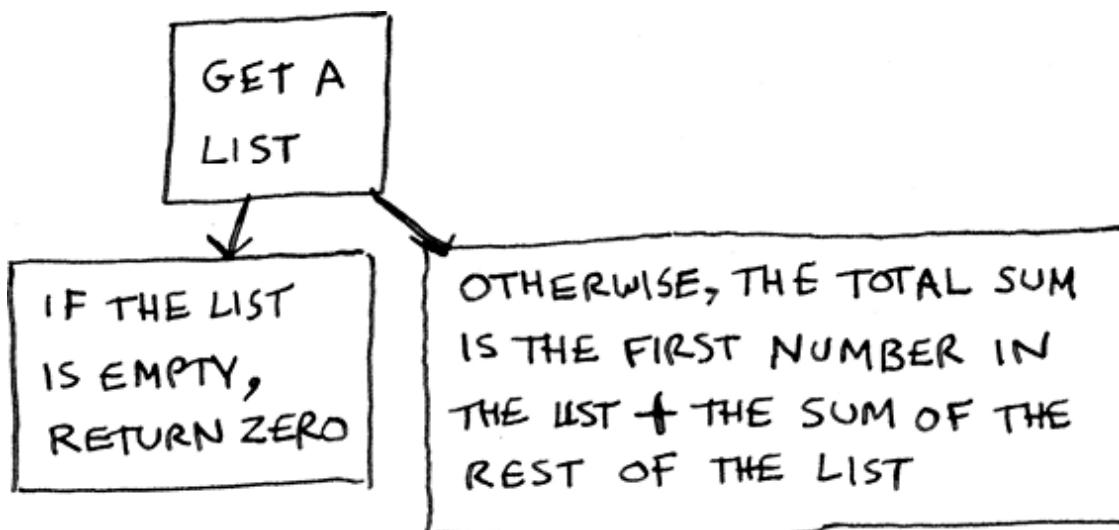
$$2 + \text{sum}([4|6]) = 2 + 1\phi = 12$$

In either case, the result is 12. But in the second version, you're passing a smaller array into the `sum` function. That is, *you decreased the size of your problem!*

En cualquier caso, el resultado es 12. Pero en la segunda versión, estás pasando una matriz más pequeña a la función `sum`. Es decir, *disminuiste el tamaño de tu problema!*

Your `sum` function could work like this.

Tu función `sum` podría funcionar así.



Here it is in action.

Aquí está en acción.

BOTH OF THESE  
ARE EQUAL

$$\rightarrow \text{sum}(\boxed{2} \boxed{4} \boxed{6})$$

$$\downarrow \\ 2 + \text{sum}(\boxed{4} \boxed{6})$$

$$\downarrow \\ 4 + \text{sum}(\boxed{6})$$

↓  
BASE CASE!  
 $\text{sum}(\boxed{6})$  is 6

FINAL RESULT

↓

12  
↑

$$2 + 10 = 12$$

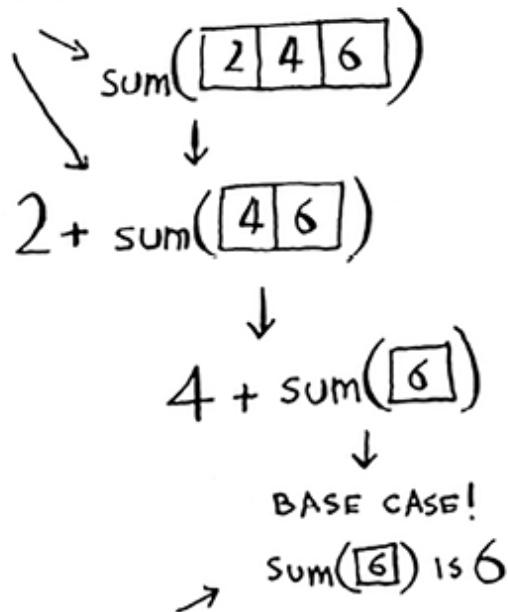
$$4 + 6 = 10$$



Remember, recursion keeps track of the state.

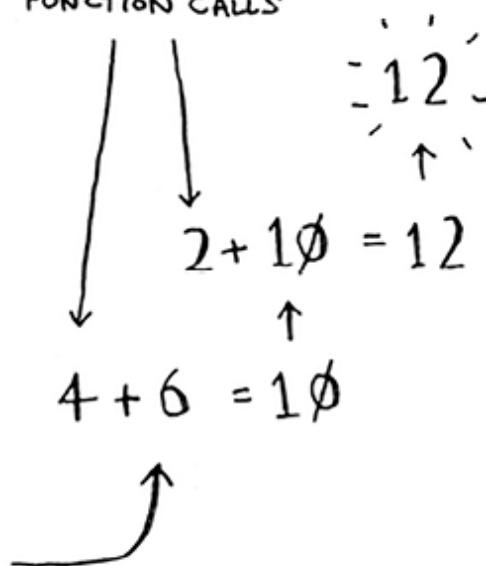
Recuerde, la recursividad realiza un seguimiento del estado.

NONE OF THESE  
FUNCTION CALLS  
COMPLETE UNTIL  
YOU HIT THE  
BASE CASE!



THIS IS THE FIRST  
FUNCTION CALL THAT  
ACTUALLY COMPLETES

REMEMBER, RECURSION  
SAVES THE STATE FOR  
THESE PARTIALLY COMPLETE  
FUNCTION CALLS



**TIP** When you're writing a recursive function involving an array, the base case is often an empty array or an array with one element. If you're stuck, try that first.

Consejo Cuando escribes una función recursiva que involucra una matriz, el caso base suele ser una matriz vacía o una matriz con un elemento. Si estás atascado, inténtalo primero.

## Sneak peek at functional programming

### Un adelanto de la programación funcional

"Why would I do this recursively if I can do it easily with a loop?" you may be thinking. Well, this is a sneak peek into functional programming! Functional programming languages like

Haskell don't have loops, so you have to use recursion to write functions like this. If you have a good understanding of recursion, functional languages will be easier to learn. For example, here's how you'd write a `sum` function in Haskell:

"¿Por qué debería hacer esto de forma recursiva si puedo hacerlo fácilmente con un bucle?" puedes estar pensando. Bueno, ¡este es un adelanto de la programación funcional! Los lenguajes de programación funcionales como Haskell no tienen bucles, por lo que debes usar la recursividad para escribir funciones como esta. Si comprende bien la recursividad, los lenguajes funcionales serán más fáciles de aprender. Por ejemplo, así es como escribirías una función `sum` en Haskell:

```
sum [] = 0          ①  
sum (x:xs) = x + (sum xs) ②
```

① Base case

① Caso base

② Recursive case

② Caso recursivo

Notice that it looks like you have two definitions for the function. The first definition runs when you hit the base case. The second definition runs at the recursive case. You can also write this function in Haskell using an `if` statement:

Observe que parece que tiene dos definiciones para la función. La primera definición se ejecuta cuando llega al caso base. La segunda definición se aplica al caso recursivo.

También puedes escribir esta función en Haskell usando una declaración `if`:

```
sum arr = if arr == []  
          then 0  
          else (head arr) + (sum (tail arr))
```

But the first definition is easier to read. Because Haskell makes heavy use of recursion, it includes all kinds of niceties like this to make recursion easy. If you like recursion or you're interested in learning a new language, check out Haskell.

Pero la primera definición es más fácil de leer. Debido a que Haskell hace un uso intensivo de la recursividad, incluye todo tipo de detalles como este para facilitar la recursividad. Si te gusta la recursividad o estás interesado en aprender un nuevo idioma, consulta Haskell.

## EXERCISES

### EJERCICIOS

**4.1** Write out the code for the earlier `sum` function.

4.1 Escriba el código para la función `sum` anterior.

**4.2** Write a recursive function to count the number of items in a list.

4.2 Escriba una función recursiva para contar el número de elementos en una lista.

**4.3** Write a recursive function to find the maximum number in a list.

4.3 Escriba una función recursiva para encontrar el número máximo en una lista.

**4.4** Remember binary search from chapter 1? It's a D&C algorithm, too. Can you come up with the base case and recursive case for binary search?

4.4 ¿Recuerdas la búsqueda binaria del capítulo 1? También es un algoritmo D&C. ¿Puedes proponer el caso base y el caso recursivo para la búsqueda binaria?



## ***Quicksort***

### ***Ordenación rápida***

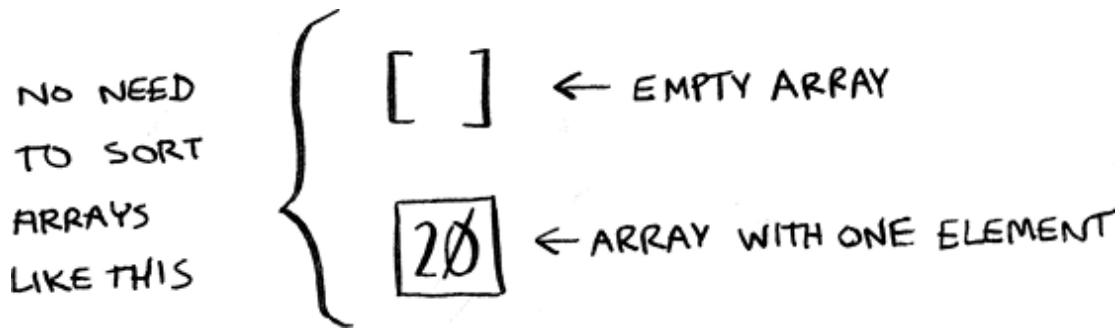
Quicksort is a sorting algorithm. It's much faster than selection sort and is frequently used in real life. Quicksort also uses D&C.

Quicksort es un algoritmo de clasificación. Es mucho más rápido que la clasificación por selección y se utiliza con frecuencia en la vida real. Quicksort también utiliza D&C.



Let's use quicksort to sort an array. What's the simplest array that a sorting algorithm can handle (remember my tip from the previous section)? Well, some arrays don't need to be sorted at all.

Usemos ordenación rápida para ordenar una matriz. ¿Cuál es la matriz más simple que puede manejar un algoritmo de clasificación (recuerde mi consejo de la sección anterior)? Bueno, algunas matrices no necesitan ordenarse en absoluto.



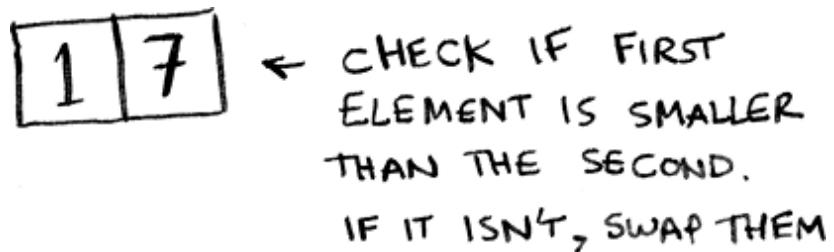
Empty arrays and arrays with just one element will be the base case. You can just return those arrays as is—there's nothing to sort:

El caso base serán los arreglos vacíos y los arreglos con un solo elemento. Puedes devolver esas matrices tal como están; no hay nada que ordenar:

```
def quicksort(array):
    if len(array) < 2:
        return array
```

Let's look at bigger arrays. An array with two elements is pretty easy to sort, too.

Veamos matrices más grandes. Una matriz con dos elementos también es bastante fácil de ordenar.



What about an array of three elements?

¿Qué pasa con una serie de tres elementos?

33	15	10
----	----	----

Remember, you're using D&C. So you want to break down this array until you're at the base case. Here's how quicksort works. First, pick an element from the array. This element is called the *pivot*.

Recuerde, está usando D&C. Entonces deseas desglosar esta matriz hasta llegar al caso base. Así es como funciona la clasificación rápida. Primero, elija un elemento de la matriz. Este elemento se llama pivote.

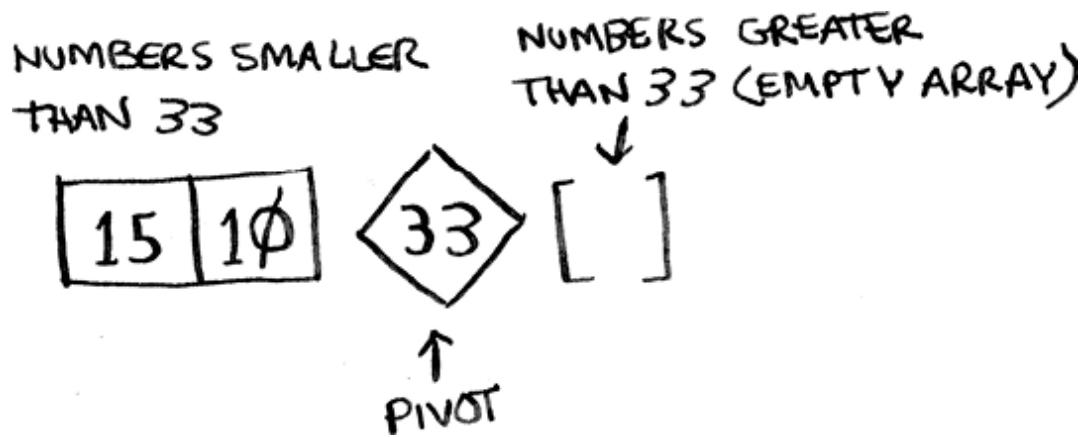


We'll talk about how to pick a good pivot later. For now, let's say the first item in the array is the pivot.

Hablaremos sobre cómo elegir un buen pivote más adelante. Por ahora, digamos que el primer elemento de la matriz es el pivote.

Now find the elements smaller than the pivot and the elements larger than the pivot.

Ahora encuentre los elementos más pequeños que el pivote y los elementos más grandes que el pivote.



This is called *partitioning*. Now you have

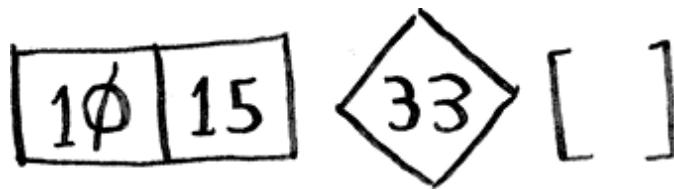
Esto se llama partición. Ahora tu tienes

- A sub-array of all the numbers less than the pivot  
Un subconjunto de todos los números menores que el pivote.
- The pivot  
el pivote
- A sub-array of all the numbers greater than the pivot  
Un subconjunto de todos los números mayores que el pivote.

The two sub-arrays aren't sorted. They're just partitioned. But if they were sorted, then sorting the whole array would be pretty easy.

Los dos subarreglos no están ordenados. Simplemente están divididos. Pero si estuvieran ordenados, entonces ordenar

todo el conjunto sería bastante fácil.



If the sub-arrays are sorted, then you can combine the whole thing as `left array + pivot + right array`, and you get a sorted array. In this case, it's  $[10, 15] + [33] + [] = [10, 15, 33]$ , which is a sorted array.

Si las submatrices están ordenadas, entonces puede combinar todo como `left array + pivot + right array` y obtendrá una matriz ordenada. En este caso, es  $[10, 15] + [33] + [] = [10, 15, 33]$ , que es una matriz ordenada.

How do you sort the sub-arrays? Well, the quicksort base case already knows how to sort empty arrays (the right sub-array), and it can recursively sort arrays of two elements (the left sub-array). So if you call quicksort on the two sub-arrays and then combine the results, you get a sorted array:

¿Cómo se ordenan las submatrices? Bueno, el caso base de ordenación rápida ya sabe cómo ordenar matrices vacías (la submatriz derecha) y puede ordenar recursivamente matrices de dos elementos (la submatriz izquierda). Entonces, si llamas a Quicksort en las dos submatrices y luego combinás los resultados, obtendrás una matriz ordenada:

```
quicksort([15, 10]) + [33] + quicksort([])  
> [10, 15, 33]
```

①

① A sorted array

① Una matriz ordenada

This strategy will work with any pivot. Suppose you choose 15 as the pivot instead.

Esta estrategia funcionará con cualquier pivote. Supongamos que elige 15 como pivote.



Both sub-arrays have only one element, and you know how to sort those. So now you know how to sort an array of three elements. Here are the steps:

Ambas submatrices tienen un solo elemento y usted sabe cómo ordenarlos. Ahora ya sabes cómo ordenar una matriz de tres elementos. Aquí están los pasos:

1. Pick a pivot.

Elige un pivote.

2. Partition the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.

Divida la matriz en dos submatrices: elementos menores que el pivote y elementos mayores que el pivote.

3. Call quicksort recursively on the two sub-arrays.

Llame a Quicksort de forma recursiva en las dos submatrices.

What about an array of four elements?

¿Qué pasa con una serie de cuatro elementos?

33	10	15	7
----	----	----	---

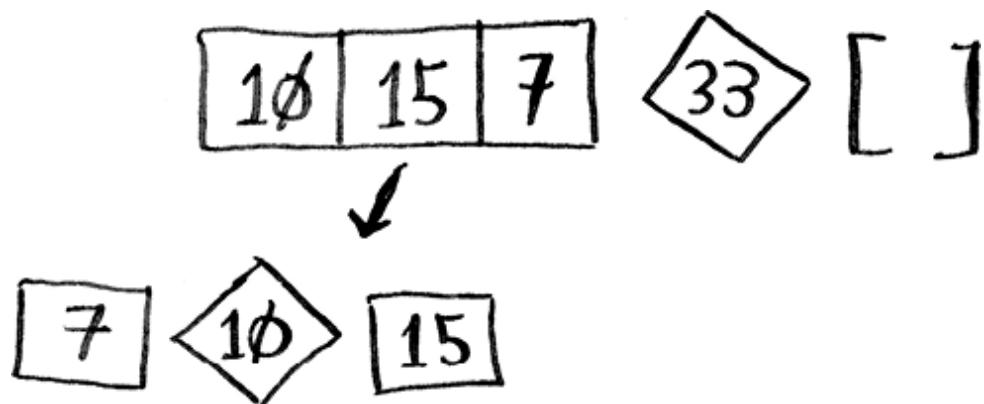
Suppose you choose 33 as the pivot again.

Supongamos que elige 33 como pivote nuevamente.

10	15	7	33	[ ]
----	----	---	----	-----

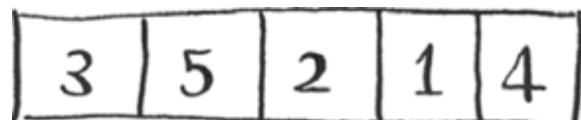
The array on the left has three elements. You already know how to sort an array of three elements: call quicksort on it recursively.

La matriz de la izquierda tiene tres elementos. Ya sabes cómo ordenar una matriz de tres elementos: llama a Quicksort de forma recursiva.



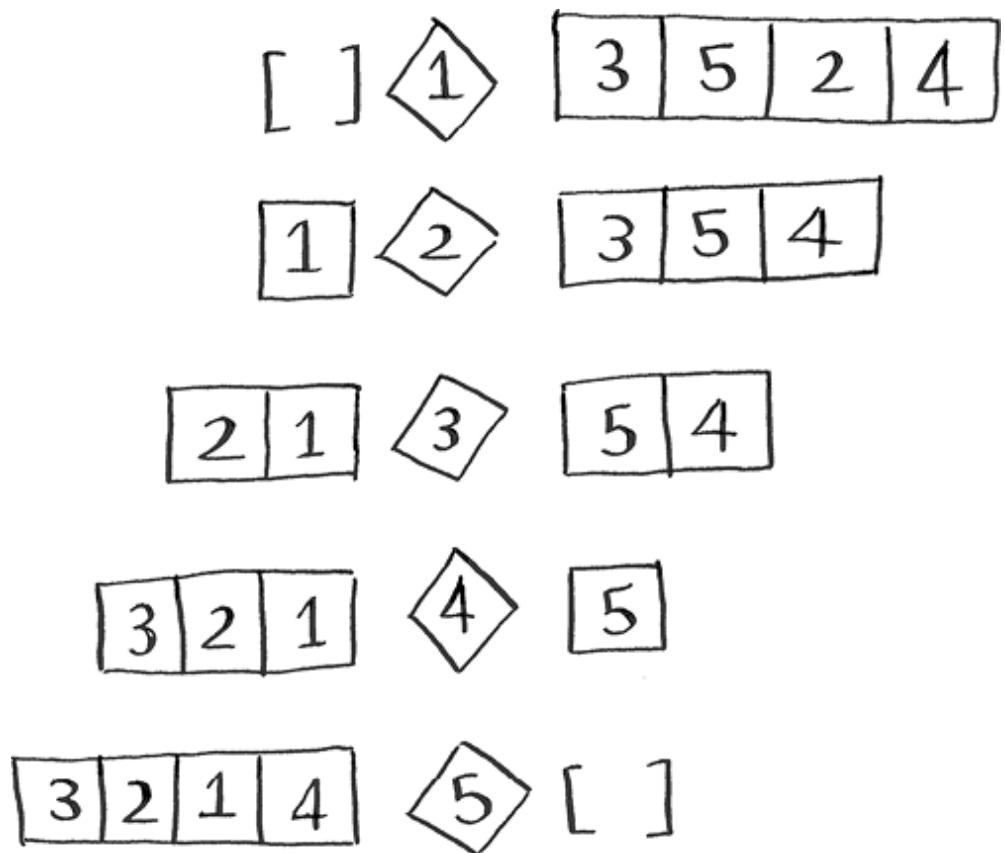
So you can sort an array of four elements. And if you can sort an array of four elements, you can sort an array of five elements. Why is that? Suppose you have this array of five elements.

Para que pueda ordenar una matriz de cuatro elementos. Y si puedes ordenar una matriz de cuatro elementos, puedes ordenar una matriz de cinco elementos. ¿Porqué es eso?  
Supongamos que tiene esta matriz de cinco elementos.



Here are all the ways you can partition this array, depending on what pivot you choose.

Aquí se muestran todas las formas en que puede particionar esta matriz, según el pivote que elija.

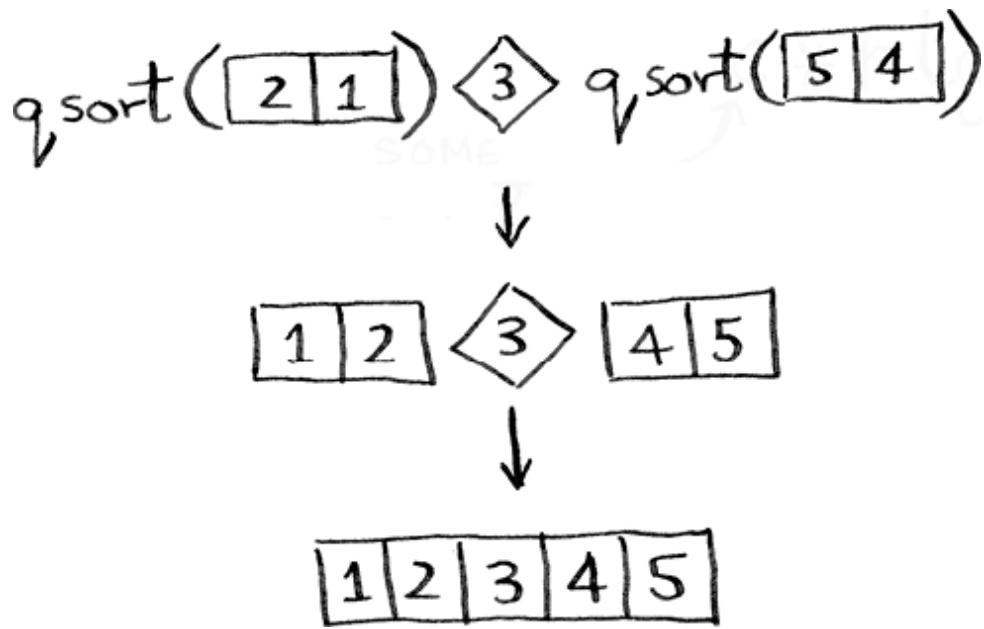


Notice that all of these sub-arrays have somewhere between zero and four elements. And you already know how to sort an array of zero to four elements using quicksort! So no matter what pivot you pick, you can call quicksort recursively on the two sub-arrays.

Observe que todos estos subarreglos tienen entre cero y cuatro elementos. ¡Y ya sabes cómo ordenar una matriz de cero a cuatro elementos usando la ordenación rápida! Entonces, no importa qué pivote elijas, puedes llamar a Quicksort de forma recursiva en los dos subarreglos.

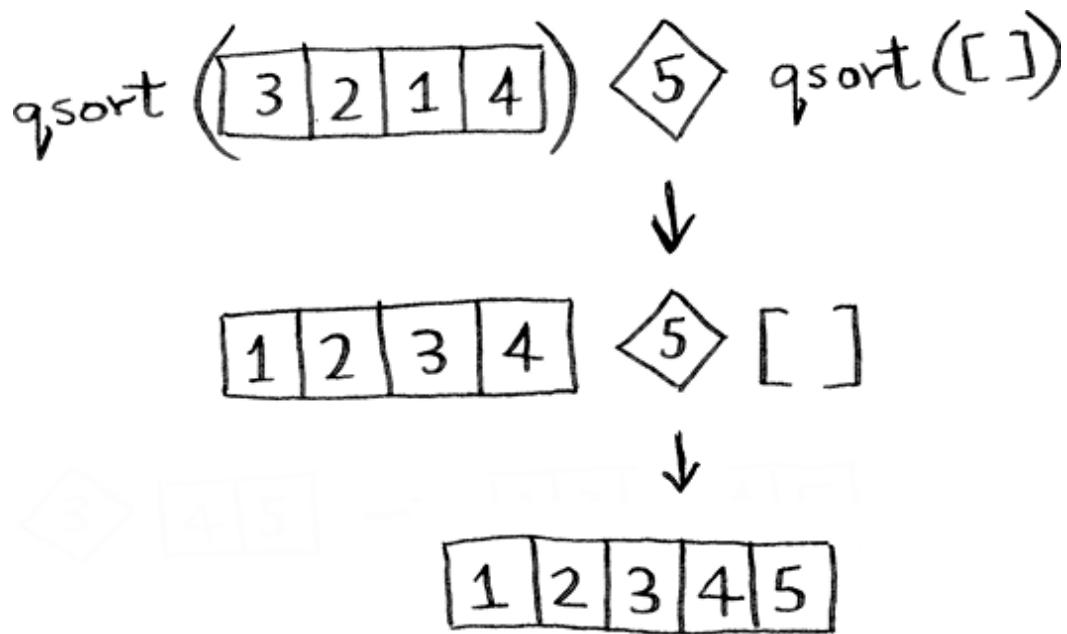
For example, suppose you pick 3 as the pivot. You call quicksort on the sub-arrays.

Por ejemplo, supongamos que elige 3 como pivote. Llama a Quicksort en las submatrices.



The sub-arrays get sorted, and then you combine the whole thing to get a sorted array. This works even if you choose 5 as the pivot.

Las submatrices se ordenan y luego se combina todo para obtener una matriz ordenada. Esto funciona incluso si eliges 5 como pivote.



This works with any element as the pivot. So you can sort an array of five elements. Using the same logic, you can sort an array of six elements and so on.

Esto funciona con cualquier elemento como pivote. Para que pueda ordenar una matriz de cinco elementos. Usando la misma lógica, puedes ordenar una matriz de seis elementos y así sucesivamente.

## Inductive proofs

### Pruebas inductivas

You just got a sneak peek into *inductive proofs!* Inductive proofs are one way to prove that your algorithm works. Each inductive proof has two steps: the base case and the inductive case. Sound familiar? For example, suppose I want to prove that I can climb to the top of a ladder. In the inductive case, if my legs are on a rung, I can put my legs on the next rung. So if I'm on rung 2, I can climb to rung 3. That's the inductive case. For the base case, I'll say

that my legs are on rung 1. Therefore, I can climb the entire ladder, going up one rung at a time.

¡Acabas de echar un vistazo a las pruebas inductivas! Las pruebas inductivas son una forma de demostrar que su algoritmo funciona. Cada prueba inductiva tiene dos pasos: el caso base y el caso inductivo. ¿Suena familiar? Por ejemplo, supongamos que quiero demostrar que puedo subir a lo alto de una escalera. En el caso inductivo, si mis piernas están en un peldaño, puedo ponerlas en el siguiente peldaño. Entonces, si estoy en el peldaño 2, puedo subir al peldaño 3. Ese es el caso inductivo. Para el caso base, diré que mis piernas están en el peldaño 1. Por lo tanto, puedo subir toda la escalera, subiendo un peldaño a la vez.

You use similar reasoning for quicksort. In the base case, I showed that the algorithm works for the base case: arrays of size 0 and 1. In the inductive case, I showed that if quicksort works for an array of size 1, it will work for an array of size 2. And if it works for arrays of size 2, it will work for arrays of size 3, and so on. Then I can say that quicksort will work for all arrays of any size. I won't go deeper into inductive proofs here, but they're fun and go hand in hand with D&C.

Utiliza un razonamiento similar para la clasificación rápida. En el caso base, demostré que el algoritmo funciona para el caso base: matrices de tamaño 0 y 1. En el caso inductivo, demostré que si la ordenación rápida funciona para una matriz de tamaño 1, funcionará para una matriz de tamaño 2. Y si funciona para matrices de tamaño 2, funcionará para matrices de tamaño 3, y así sucesivamente. Entonces puedo decir que la ordenación rápida funcionará para todas las matrices de cualquier tamaño. No profundizaré aquí en las pruebas inductivas, pero son divertidas y van de la mano con D&C.



Here's the code for quicksort:

Aquí está el código para la clasificación rápida:

```
def quicksort(array):
    if len(array) < 2:
        return array
    else:
        pivot = array[0]
        less = [i for i in array[1:] if i <= pivot]
        greater = [i for i in array[1:] if i > pivot]
        return quicksort(less) + [pivot] + quicksort(greater)

print(quicksort([10, 5, 2, 3]))
```

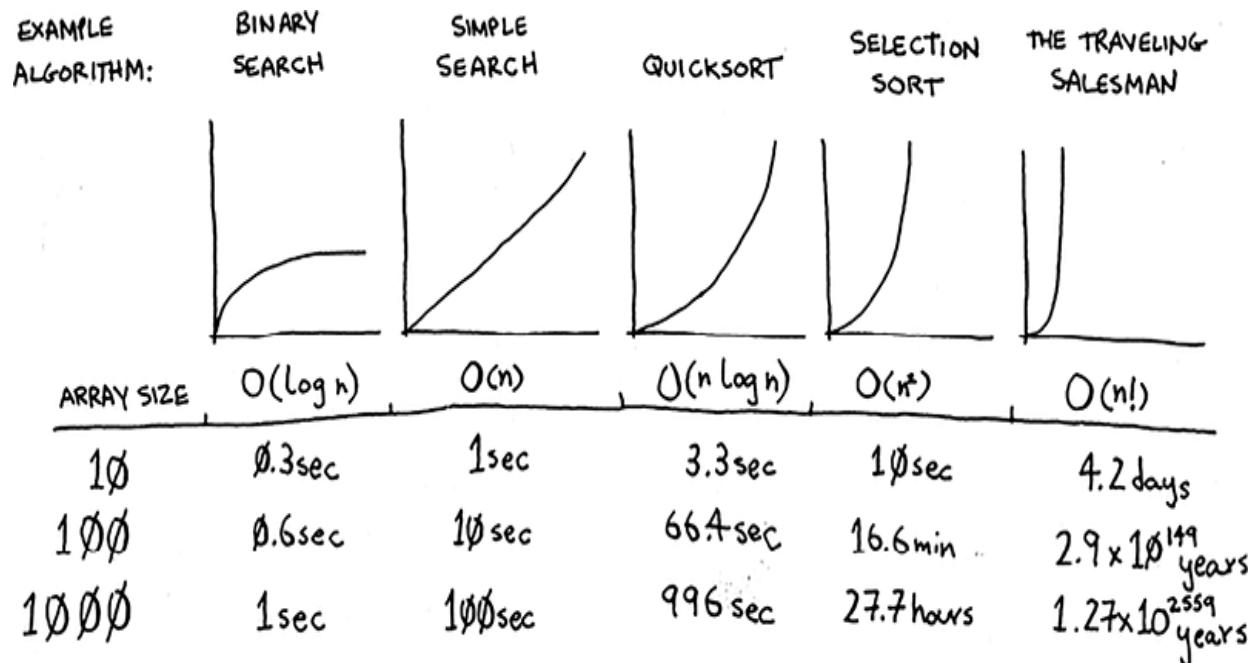
- ① Base case: arrays with 0 or 1 element are already sorted.
- ① Caso base: las matrices con 0 o 1 elemento ya están ordenadas.
- ② Recursive case
- ② Caso recursivo
- ③ Sub-array of all the elements less than the pivot
- ③ Subconjunto de todos los elementos menores que el pivote
- ④ Sub-array of all the elements greater than the pivot
- ④ Subconjunto de todos los elementos mayores que el pivote

## ***Big O notation revisited***

### ***Notación Big O revisada***

Quicksort is unique because its speed depends on the pivot you choose. Before I talk about quicksort, let's look at the most common big O run times again.

Quicksort es único porque su velocidad depende del pivote que elijas. Antes de hablar sobre la ordenación rápida, veamos nuevamente los tiempos de ejecución de O grande más comunes.



**Estimates based on a slow computer that performs 10 operations per second**

**Estimaciones basadas en una computadora lenta que realiza 10 operaciones por segundo**

The example times in this chart are estimates if you perform 10 operations per second. These graphs aren't precise—they're just there to give you a sense of how different these run times are. In reality, your computer can do way more than 10 operations per second.

Los tiempos de ejemplo en este cuadro son estimaciones si realiza 10 operaciones por segundo. Estos gráficos no son precisos; solo están ahí para darle una idea de cuán diferentes son estos tiempos de ejecución. En realidad, su computadora puede realizar más de 10 operaciones por segundo.

Each run time also has an example algorithm attached. Check out selection sort, which you learned in chapter 2. It's  $O(n^2)$ . That's a pretty slow algorithm.

Cada tiempo de ejecución también tiene un algoritmo de ejemplo adjunto. Consulte la clasificación por selección, que aprendió en el capítulo 2. Es  $O(n^2)$ . Ese es un algoritmo bastante lento.

There's another sorting algorithm called *merge sort*, which is  $O(n \log n)$ . Much faster! Quicksort is a tricky case. In the worst case, quicksort takes  $O(n^2)$  time.

Hay otro algoritmo de clasificación llamado clasificación por combinación, que es  $O(n \log n)$ . ¡Mucho más rápido! Quicksort es un caso complicado. En el peor de los casos, la ordenación rápida lleva  $O(n^2)$  tiempo.

It's as slow as selection sort! But that's the worst case. In the average case, quicksort takes  $O(n \log n)$  time. So you might be wondering:

¡Es tan lento como ordenar por selección! Pero ese es el peor de los casos. En el caso promedio, la clasificación rápida lleva  $O(n \log n)$  tiempo. Entonces quizás te preguntes:

What do *worst case* and *average case* mean here?

¿Qué significan aquí el peor de los casos y el caso promedio?

If quicksort is  $O(n \log n)$  on average, but merge sort is  $O(n \log n)$  always, why not use merge sort? Isn't it faster?

Si la clasificación rápida es  $O(n \log n)$  en promedio, pero la clasificación por combinación es  $O(n \log n)$  siempre, ¿por qué no utilizar la clasificación por combinación? ¿No es más rápido?

## Merge sort vs. quicksort

### Combinar clasificación versus clasificación rápida

Suppose you have this simple function to print every item in a list:

Suponga que tiene esta función simple para imprimir cada elemento de una lista:

```
def print_items(myList):
    for item in myList:
        print(item)
```

This function goes through every item in the list and prints it out. Because it loops over the whole list once, this function runs in  $O(n)$  time. Now, suppose you change this function so it sleeps for 1 second before it prints out an item:

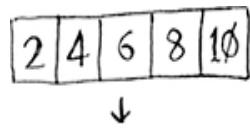
Esta función revisa cada elemento de la lista y lo imprime. Debido a que recorre toda la lista una vez, esta función se ejecuta en tiempo  $O(n)$ . Ahora, supongamos que cambia esta función para que entre en suspensión durante 1 segundo antes de imprimir un elemento:

```
from time import sleep
def print_items2(myList):
    for item in myList:
```

```
sleep(1)  
print(item)
```

Before it prints out an item, it will pause for 1 second. Suppose you print a list of five items using both functions.

Antes de imprimir un elemento, hará una pausa de 1 segundo. Suponga que imprime una lista de cinco elementos usando ambas funciones.



print\_items: 2 4 6 8 10

print\_items2: <sleep> 2 <sleep> 4 <sleep> 6 <sleep> 8 <sleep> 10

Both functions loop through the list once, so they're both  $O(n)$  time. Which one do you think will be faster in practice? I think `print_items` will be much faster because it doesn't pause for 1 second before printing an item. So even though both functions are the same speed in big O notation, `print_items` is faster in practice. When you write big O notation like  $O(n)$ , it really means this.

Ambas funciones recorren la lista una vez, por lo que ambas son tiempo  $O(n)$ . ¿Cuál crees que será más rápido en la práctica? Creo que `print_items` será mucho más rápido porque no se detiene durante 1 segundo antes de imprimir un elemento. Entonces, aunque ambas funciones tienen la misma velocidad en notación  $O$  grande, `print_items` es más

rápido en la práctica. Cuando escribes notación O grande como  $O(n)$ , realmente significa esto.

*C \* h*  
↑  
SOME  
FIXED AMOUNT  
OF TIME

$c$  is some fixed amount of time that your algorithm takes. It's called the *constant*. For example, it might be 10 milliseconds \*  $n$  for `print_items` versus 1 second \*  $n$  for `print_items2`.

$c$  es una cantidad fija de tiempo que tarda su algoritmo. Se llama constante. Por ejemplo, podría ser 10 milliseconds \*  $n$  para `print_items` versus 1 second \*  $n$  para `print_items2`.

You usually ignore that constant because if two algorithms have different big O times, the constant doesn't matter. Take binary search and simple search, for example. Suppose both algorithms had these constants.

Por lo general, se ignora esa constante porque si dos algoritmos tienen tiempos O grandes diferentes, la constante no importa. Tomemos como ejemplo la búsqueda binaria y la búsqueda simple. Supongamos que ambos algoritmos tuvieran estas constantes.

$$\frac{10 \text{ ms} * n}{\text{SIMPLE SEARCH}}$$

$$\frac{1 \text{ sec} * \log n}{\text{BINARY SEARCH}}$$

You might say, "Wow! Simple search has a constant of 10 ms, but binary search has a constant of 1 second. Simple search is way faster!" Now suppose you're searching a list of 4 billion elements. Here are the times.

Podrías decir: "¡Guau! La búsqueda simple tiene una constante de 10 ms, pero la búsqueda binaria tiene una constante de 1 segundo. ¡La búsqueda simple es mucho más rápida! Ahora suponga que está buscando una lista de 4 mil millones de elementos. Aquí están los tiempos.

SIMPLE SEARCH	$10 \text{ ms} * 4 \text{ BILLION}$	=	463 days
BINARY SEARCH	$1 \text{ sec} * 32$	=	32 seconds

We're using 32 for binary search because binary search runs in log time and  $\log(4 \text{ billion})$  equals 32. As you can see, binary search is still way faster. That constant didn't make a difference at all.

Usamos 32 para la búsqueda binaria porque la búsqueda binaria se ejecuta en tiempo de registro y  $\log(4 \text{ mil millones})$  equivale a 32. Como puede ver, la búsqueda binaria sigue siendo mucho más rápida. Esa constante no hizo ninguna diferencia.

But sometimes the constant *can* make a difference. Quicksort versus merge sort is one example. Often, given the way quicksort and merge sort are implemented, if they're both  $O(n \log n)$  time, quicksort is faster. And quicksort is faster in practice because it hits the average case way more often than the worst case.

Pero a veces la constante puede marcar la diferencia. La clasificación rápida versus la clasificación por combinación es un ejemplo. A menudo, dada la forma en que se implementan la ordenación rápida y la ordenación por combinación, si ambas son  $O(n \log n)$  tiempo, la ordenación rápida es más rápida. Y la clasificación rápida es más rápida en la práctica porque llega al caso promedio con mucha más frecuencia que al peor de los casos.

So now you're wondering: What's the average case versus the worst case?

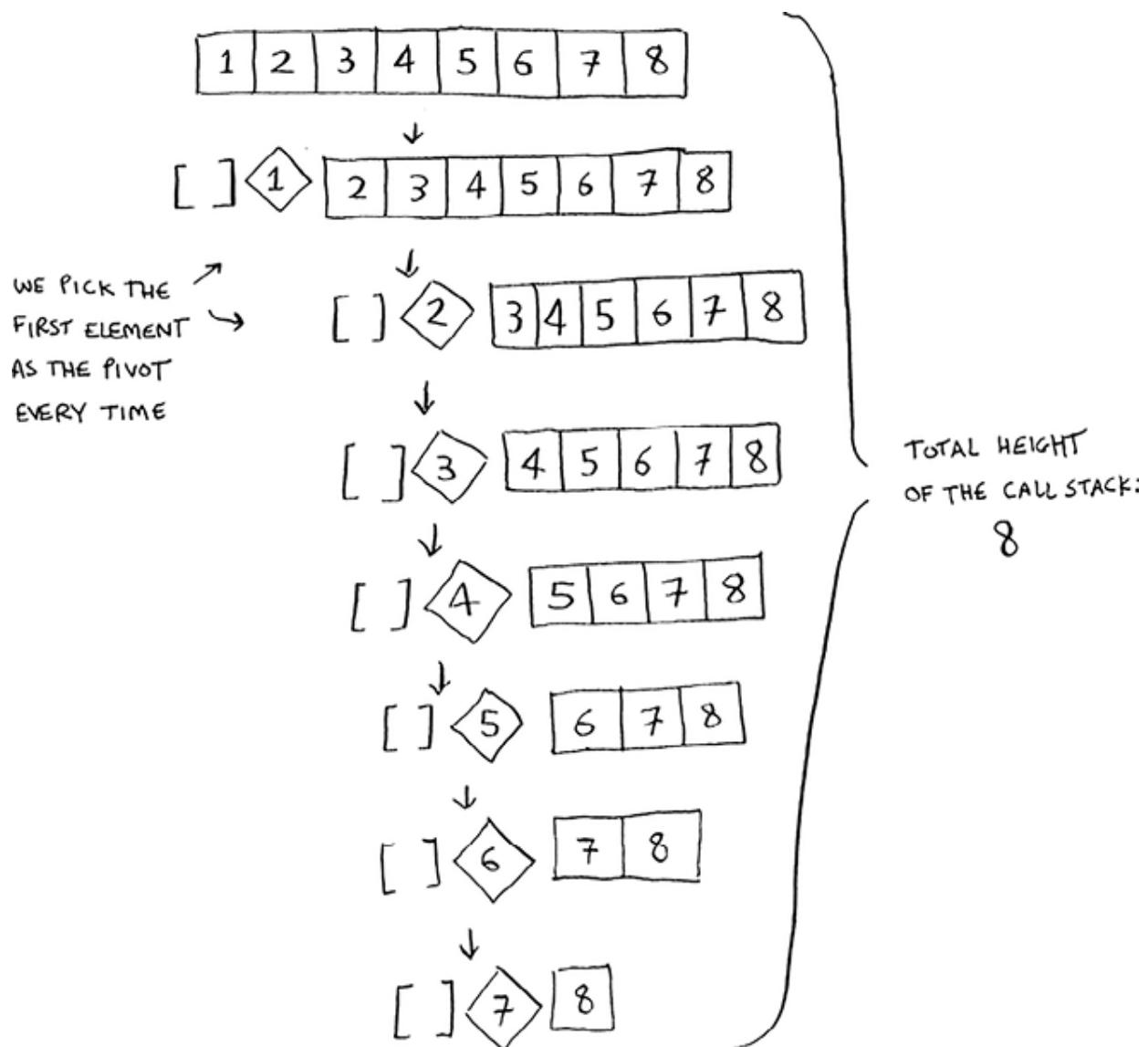
Entonces ahora te preguntas: ¿Cuál es el caso promedio versus el peor de los casos?

## Average case vs. worst case

### Caso promedio versus peor caso

The performance of quicksort heavily depends on the pivot you choose. Suppose you always choose the first element as the pivot. And you call quicksort with an array that is *already sorted*. Quicksort doesn't check to see whether the input array is already sorted. So it will still try to sort it.

El rendimiento de Quicksort depende en gran medida del pivote que elija. Supongamos que siempre elige el primer elemento como pivote. Y llamas a Quicksort con una matriz que ya está ordenada. Quicksort no verifica si la matriz de entrada ya está ordenada. Por lo tanto, seguirá intentando ordenarlo.

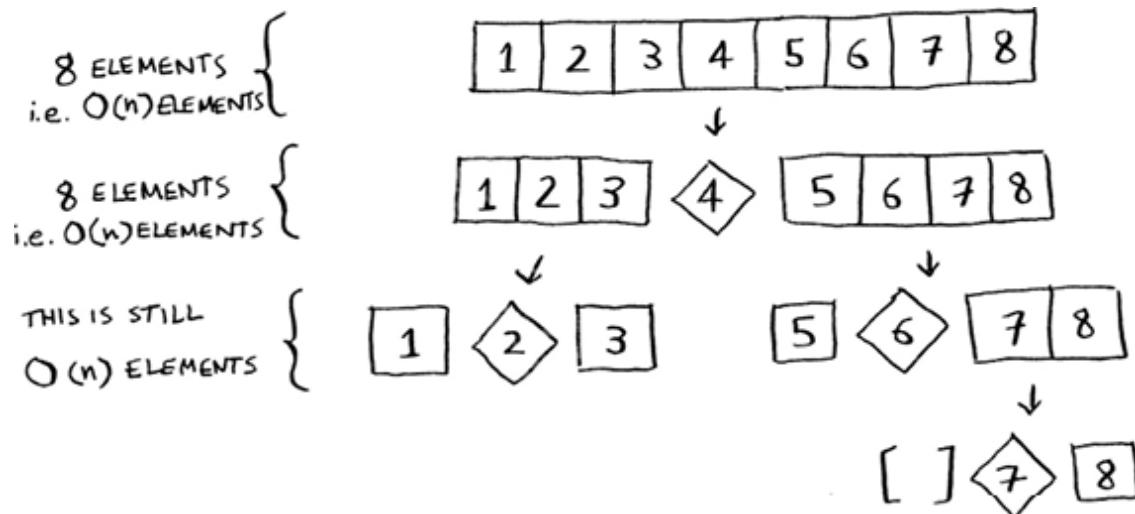


Notice that you're not splitting the array into two halves. Instead, one of the sub-arrays is always empty. So the call

stack is really long. Now, instead, suppose you always picked the middle element as the pivot. Look at the call stack now.

Observe que no está dividiendo la matriz en dos mitades. En cambio, una de las submatrices siempre está vacía.

Entonces la pila de llamadas es realmente larga. Ahora, en cambio, supongamos que siempre eliges el elemento central como pivote. Mire la pila de llamadas ahora.



It's so short! Because you divide the array in half every time, you don't need to make as many recursive calls. You hit the base case sooner, and the call stack is much shorter.

¡Es tan corto! Debido a que divide la matriz por la mitad cada vez, no necesita realizar tantas llamadas recursivas. Se llega al caso base antes y la pila de llamadas es mucho más corta.

The first example you saw is the worst-case scenario, and the second example is the best-case scenario. In the worst

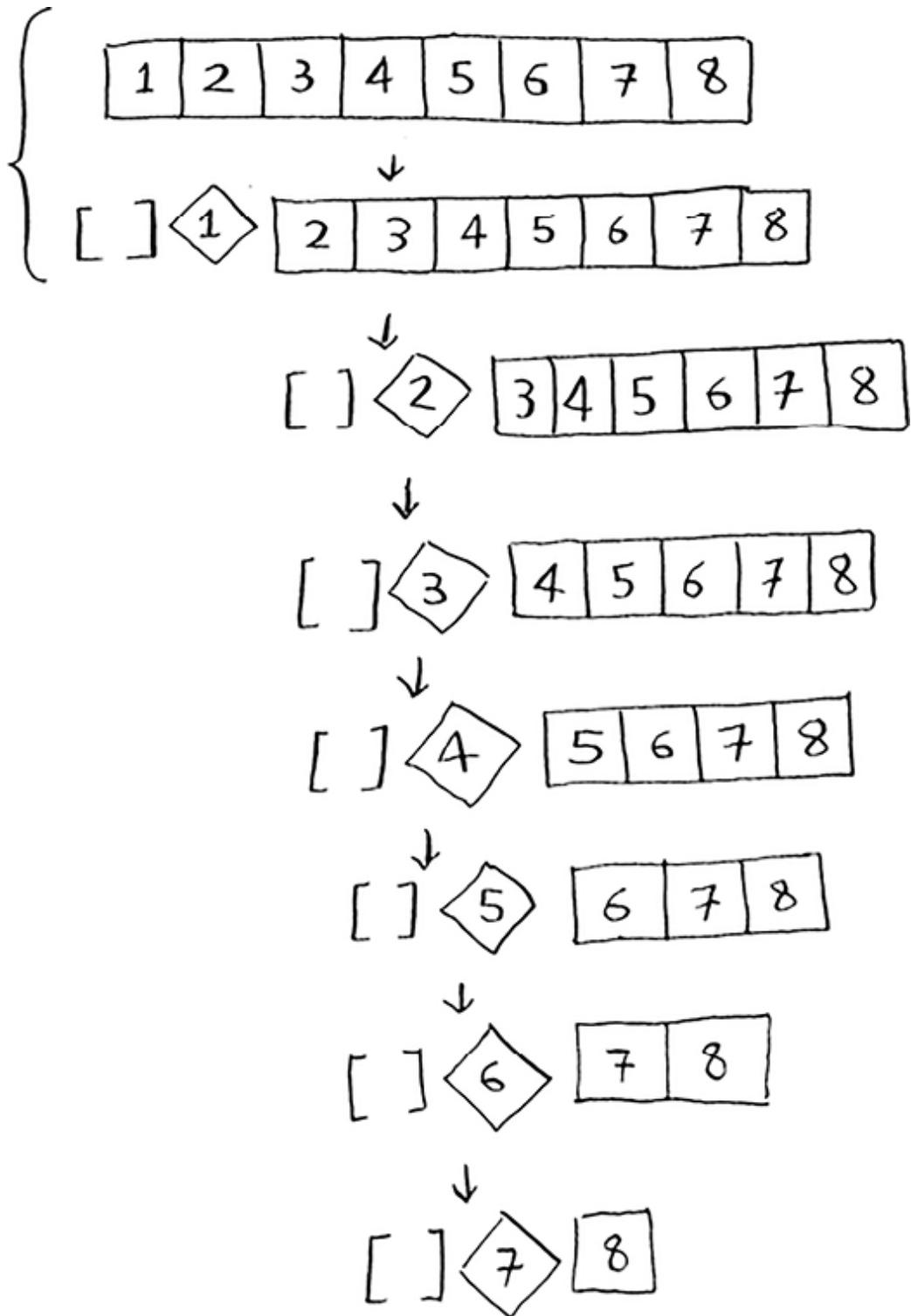
case, the stack size is  $O(n)$ . In the best case, the stack size is  $O(\log n)$ . Read on to find out how this affects the worst and average case running times.

El primer ejemplo que vio es el peor de los casos y el segundo ejemplo es el mejor de los casos. En el peor de los casos, el tamaño de la pila es  $O(n)$ . En el mejor de los casos, el tamaño de la pila es  $O(\log n)$ . Continúe leyendo para descubrir cómo esto afecta los tiempos de ejecución de los casos peores y promedio.

Look at the first level in the stack. You pick one element as the pivot, and the rest of the elements are divided into subarrays. You touch all eight elements in the array. So this first operation takes  $O(n)$  time. You touched all eight elements on this level of the call stack. But actually, you touch  $O(n)$  elements on every level of the call stack.

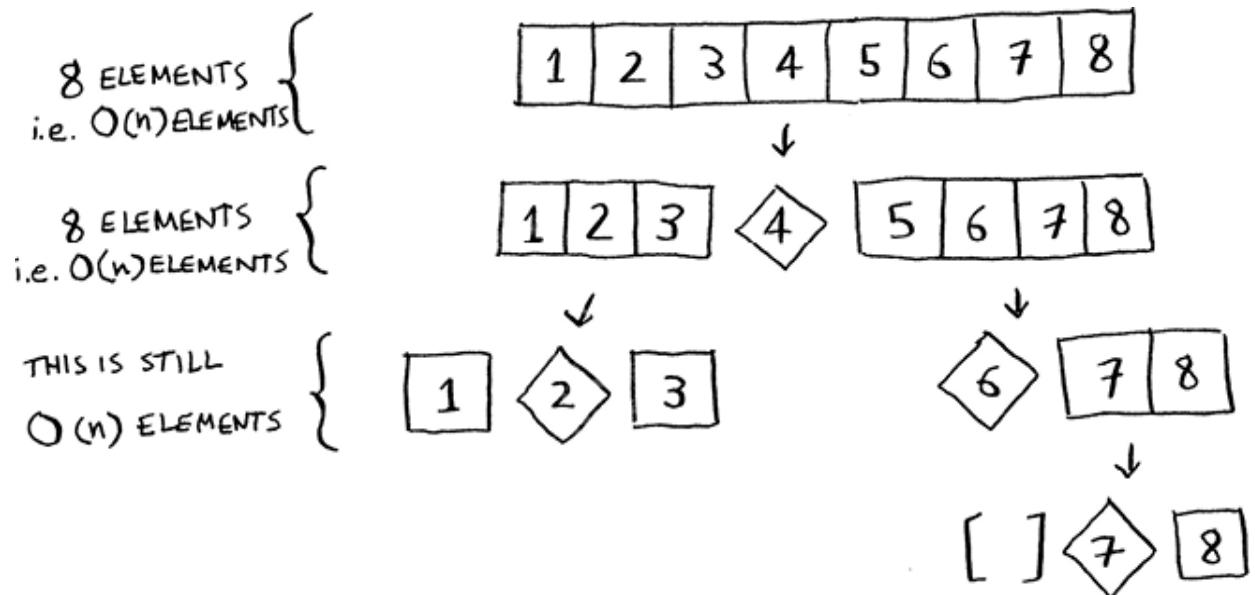
Mira el primer nivel de la pila. Elige un elemento como pivote y el resto de los elementos se dividen en submatrices. Tocas los ocho elementos de la matriz. Entonces esta primera operación toma  $O(n)$  tiempo. Tocó los ocho elementos en este nivel de la pila de llamadas. Pero en realidad, tocas elementos  $O(n)$  en cada nivel de la pila de llamadas.

BOTH OF  
THESE ARE  
 $O(n)$   
ELEMENTS



Even if you partition the array differently, you're still touching  $O(n)$  elements every time.

Incluso si divide la matriz de manera diferente, seguirá tocando elementos  $O(n)$  cada vez.



In this example, there are  $O(\log n)$  levels (the technical way to say that is, "The height of the call stack is  $O(\log n)$ "). And each level takes  $O(n)$  time. The entire algorithm will take  $O(n) * O(\log n) = O(n \log n)$  time. This is the best-case scenario.

En este ejemplo, hay niveles  $O(\log n)$  (la forma técnica de decirlo es: "La altura de la pila de llamadas es  $O(\log n)$ "). Y cada nivel lleva  $O(n)$  tiempo. Todo el algoritmo tomará  $O(n) * O(\log n) = O(n \log n)$  tiempo. Este es el mejor de los casos.

In the worst case, there are  $O(n)$  levels, so the algorithm will take  $O(n) * O(n) = O(n^2)$  time.

En el peor de los casos, hay niveles  $O(n)$ , por lo que el algoritmo tardará  $O(n) * O(n) = O(n^2)$  tiempo.

Well, guess what? I'm here to tell you that the best case is also the average case. *If you always choose a random element in the array as the pivot*, quicksort will complete in  $O(n \log n)$  time on average. (There's one exception: if all the elements in your array are the same, you will always hit the worst-case run time without some additional logic.)

¿Bien adivina que? Estoy aquí para decírselos que el mejor de los casos es también el caso promedio. Si siempre elige un elemento aleatorio en la matriz como pivote, la clasificación rápida se completará en un tiempo promedio de  $O(n \log n)$ . (Hay una excepción: si todos los elementos de su matriz son iguales, siempre alcanzará el peor tiempo de ejecución sin alguna lógica adicional).

Quicksort is one of the fastest sorting algorithms out there, and it's a very good example of D&C.

Quicksort es uno de los algoritmos de clasificación más rápidos que existen y es un muy buen ejemplo de D&C.

## EXERCISES

### EJERCICIOS

How long would each of these operations take in big O notation?

¿Cuánto tiempo tomaría cada una de estas operaciones en notación O grande?

**4.5** Printing the value of each element in an array.

4.5 Imprimir el valor de cada elemento de un array.

**4.6** Doubling the value of each element in an array.

4.6 Duplicar el valor de cada elemento de una matriz.

**4.7** Doubling the value of just the first element in an array.

4.7 Duplicar el valor de solo el primer elemento de una matriz.

**4.8** Creating a multiplication table with all the elements in the array. So if your array is [2, 3, 7, 8, 10], you first multiply every element by 2, then multiply every element by 3, then by 7, and so on.

4.8 Creando una tabla de multiplicar con todos los elementos de la matriz. Entonces, si tu matriz es [2, 3, 7, 8, 10], primero multiplicas cada elemento por 2, luego multiplicas cada elemento por 3, luego por 7, y así sucesivamente.

	2	3	7	8	10
2	4	6	14	16	20
3	6	9	21	24	30
7	14	21	49	56	70
8	16	24	56	64	80
10	20	30	70	80	100

## Recap

## Resumen

- D&C works by breaking a problem down into smaller and smaller pieces. If you're using D&C on a list, the base case is probably an empty array or an array with one element.

D&C funciona dividiendo un problema en partes cada vez más pequeñas. Si está utilizando D&C en una lista, el caso base probablemente sea una matriz vacía o una matriz con un elemento.

- If you're implementing quicksort, choose a random element as the pivot. The average run time of quicksort is  $O(n \log n)$ !

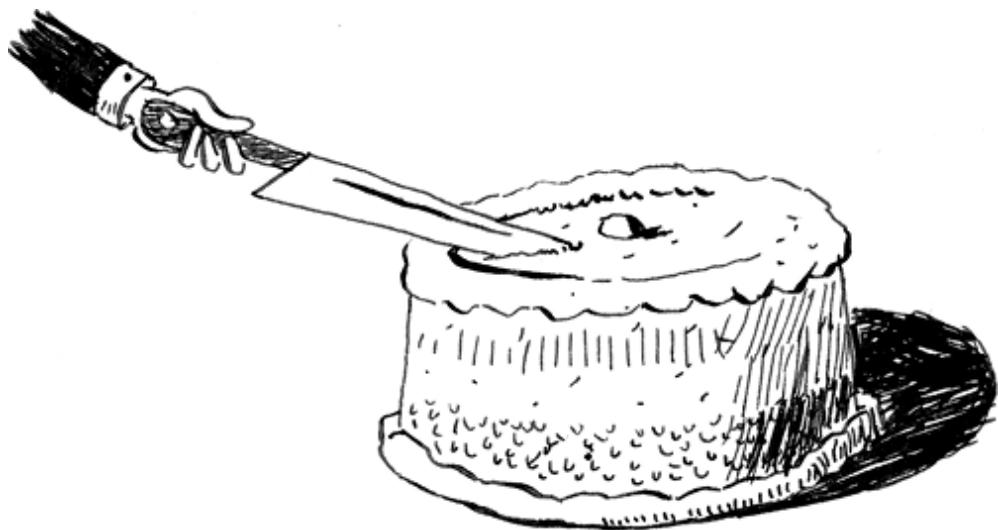
Si está implementando la ordenación rápida, elija un elemento aleatorio como pivote. ¡El tiempo de ejecución promedio de Quicksort es  $O(n \log n)$ !

- Given two algorithms with the same big O running time, one can be consistently faster than the other. That's why quicksort is faster than merge sort.

Dados dos algoritmos con el mismo tiempo de ejecución de O grande, uno puede ser consistentemente más rápido que el otro. Es por eso que la clasificación rápida es más rápida que la clasificación por combinación.

- The constant almost never matters for simple search versus binary search because  $O(\log n)$  is so much faster than  $O(n)$  when your list gets big.

La constante casi nunca importa para la búsqueda simple versus la búsqueda binaria porque  $O(\log n)$  es mucho más rápido que  $O(n)$  cuando la lista crece.



# **5 Hash tables**

---

## **5 tablas hash**

---

### **In this chapter**

#### **En este capítulo**

- You learn about hash tables, one of the most useful data structures. Hash tables have many uses; this chapter covers the common use cases.

Aprenderá sobre las tablas hash, una de las estructuras de datos más útiles. Las tablas hash tienen muchos usos; Este capítulo cubre los casos de uso comunes.

- You learn about the internals of hash tables: implementation, collisions, and hash functions. These properties will help you understand how to analyze a hash table's performance.

Aprenderá sobre los aspectos internos de las tablas hash: implementación, colisiones y funciones hash. Estas propiedades le ayudarán a comprender cómo analizar el rendimiento de una tabla hash.

Suppose you work at a grocery store. When a customer buys produce, you have to look up the price in a book. If the book is unalphabetized, it can take you a long time to look through every single line for *apple*. You'd be doing simple search from chapter 1, where you have to look at every line. Do you remember how long that would take?  $O(n)$  time. If

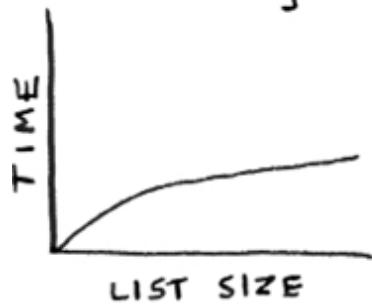
the book is alphabetized, you could run binary search to find the price of an apple. That would only take  $O(\log n)$  time.

Supongamos que trabaja en una tienda de comestibles. Cuando un cliente compra un producto, hay que buscar el precio en un libro. Si el libro no está alfabético, puede llevarle mucho tiempo revisar cada línea de Apple. Estarías haciendo una búsqueda simple desde el capítulo 1, donde debes mirar cada línea. ¿Recuerdas cuánto tiempo tomaría eso? A tiempo. Si el libro está ordenado alfabéticamente, puedes ejecutar una búsqueda binaria para encontrar el precio de una manzana. Eso sólo tomaría  $O(\log n)$  tiempo.



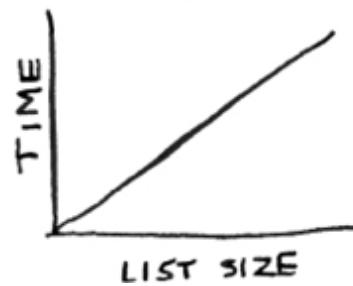
EGGS... 2.49\$  
MILK.... 1.99\$  
PEAR.... 79¢

SORTED LIST  
 $O(\log n)$



PEAR.. 79¢  
EGGS... 2.49\$  
MILK.... 1.99\$

UNSORTED LIST  
 $O(n)$



As a reminder, there's a big difference between  $O(n)$  and  $O(\log n)$  time! Suppose you could look through 10 lines of the book per second. Here's how long simple search and binary search would take you.

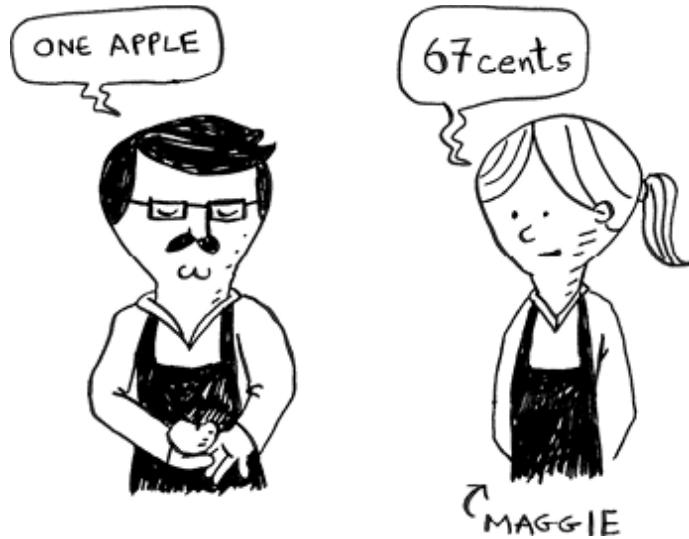
Como recordatorio, ¡hay una gran diferencia entre el tiempo  $O(n)$  y  $O(\log n)$ ! Supongamos que pudieras leer 10 líneas del

libro por segundo. A continuación le indicamos cuánto tiempo le llevaría una búsqueda simple y una búsqueda binaria.

# OF ITEMS IN THE BOOK	$O(n)$	$O(\log n)$
100	10 sec	1 sec ← YOU NEED TO CHECK $\log_2 100 = 7$ LINES
1000	1.66 min	1 sec ← NEED TO CHECK $\log_2 1000 = 10$ LINES
10000	16.6 min	2 sec ← $\log_2 10000 = 14$ LINES = 2 sec

You already know that binary search is darn fast. But as a cashier, looking things up in a book is a pain, even if the book is sorted. You can feel the customer steaming up as you search for items in the book. What you really need is a buddy who has all the names and prices memorized. Then you don't need to look up anything: you ask her, and she tells you the answer instantly.

Ya sabes que la búsqueda binaria es increíblemente rápida. Pero como cajero, buscar cosas en un libro es una molestia, incluso si el libro está ordenado. Puede sentir que el cliente se enfurece mientras busca artículos en el libro. Lo que realmente necesitas es un amigo que tenga todos los nombres y precios memorizados. Entonces no hace falta que busques nada: le preguntas y ella te dice la respuesta al instante.



Your buddy Maggie can give you the price in  $O(1)$  time for any item, no matter how big the book is. She's even faster than binary search.

Tu amiga Maggie puede darte el precio en  $O(1)$  tiempo de cualquier artículo, sin importar cuán grande sea el libro. Es incluso más rápida que la búsqueda binaria.

# OF ITEMS IN THE BOOK	SIMPLE SEARCH	BINARY SEARCH	MAGGIE
100	$O(n)$	$O(\log n)$	$O(1)$
1000	10 sec	1 sec	INSTANT
10000	1.6 min	1 sec	INSTANT
100000	16.6 min	2 sec	INSTANT

What a wonderful person! How do you get a "Maggie"?

iQué persona tan maravillosa! ¿Cómo se consigue una "Maggie"?

Let's put on our data structure hats. You know two data structures so far: arrays and lists (I won't talk about stacks because you can't really "search" for something in a stack). You could implement this book as an array.

Pongámonos el sombrero de estructura de datos. Hasta ahora conoces dos estructuras de datos: matrices y listas (no hablaré de pilas porque realmente no puedes "buscar" algo en una pila). Podrías implementar este libro como una matriz.

(EGGS, 2.49)	(MILK, 1.49)	(PEAR, 0.79)
--------------	--------------	--------------

Each item in the array is really two items: one is the name of a kind of produce, and the other is the price. If you sort this array by name, you can run binary search on it to find the price of an item. So you can find items in  $O(\log n)$  time. But you want to find items in  $O(1)$  time. That is, you want to make a "Maggie." That's where hash functions come in.

Cada elemento de la matriz consta en realidad de dos elementos: uno es el nombre de un tipo de producto y el otro es el precio. Si ordena esta matriz por nombre, puede ejecutar una búsqueda binaria para encontrar el precio de un artículo. Para que pueda encontrar elementos en tiempo  $O(\log n)$ . Pero desea encontrar elementos en tiempo  $O(1)$ . Es

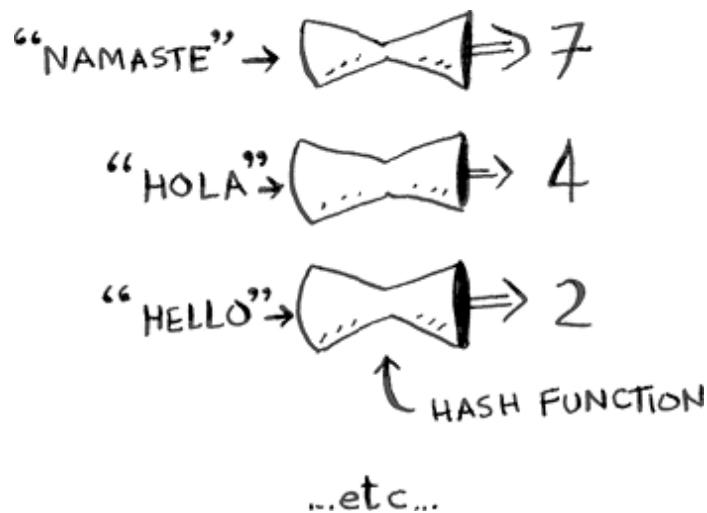
dicho, quieres hacer una "Maggie". Ahí es donde entran las funciones hash.

## Hash functions

### Funciones hash

A hash function is a function where you put in a string (*string* here means any kind of data—a sequence of bytes), and you get back a number.

Una función hash es una función en la que ingresas una cadena (aquí cadena significa cualquier tipo de datos, una secuencia de bytes) y obtienes un número.



In technical terminology, we'd say that a hash function "maps strings to numbers." You might think there's no discernable pattern to what number you get out when you put a string in. But there are some requirements for a hash function:

En terminología técnica, diríamos que una función hash "asigna cadenas a números". Podría pensar que no existe un patrón discernible sobre el número que obtiene cuando ingresa una cadena. Pero existen algunos requisitos para una función hash:

- It needs to be consistent. For example, suppose you put in "apple" and get back "3." Every time you put in "apple," you should get "3" back. Without this, your hash table won't work.

Tiene que ser consistente. Por ejemplo, supongamos que ingresa "manzana" y obtiene "3". Cada vez que pongas "manzana", deberías obtener "3" de vuelta. Sin esto, su tabla hash no funcionará.

- It should map different words to different numbers. For example, a hash function is no good if it always returns "1" for any word you put in. In the best case, every different word should map to a different number.

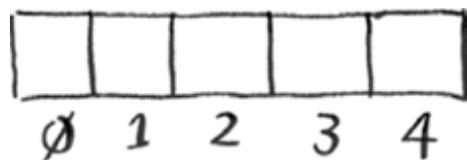
Debería asignar diferentes palabras a diferentes números. Por ejemplo, una función hash no sirve si siempre devuelve "1" para cualquier palabra que ingrese. En el mejor de los casos, cada palabra diferente debería asignarse a un número diferente.

So a hash function maps strings to numbers. What is that good for? Well, you can use it to make your "Maggie"!

Entonces, una función hash asigna cadenas a números. ¿Para qué sirve eso? Bueno, puedes usarlo para hacer tu "Maggie"!

Start with an empty array:

Comience con una matriz vacía:



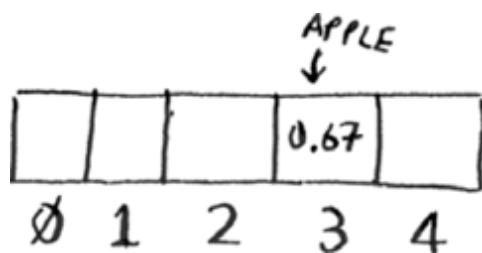
You'll store all of your prices in this array. Let's add the price of an apple. Feed "apple" into the hash function.

Almacenará todos sus precios en esta matriz. Sumemos el precio de una manzana. Introduzca "manzana" en la función hash.



The hash function outputs "3." So let's store the price of an apple at index 3 in the array.

La función hash genera "3". Entonces almacenemos el precio de una manzana en el índice 3 de la matriz.



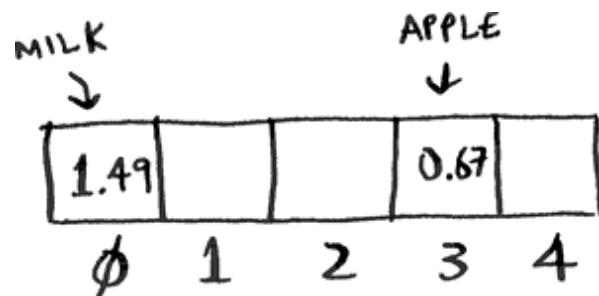
Let's add milk. Feed "milk" into the hash function.

Agreguemos leche. Introduzca "leche" en la función hash.



The hash function says "0." Let's store the price of milk at index 0.

La función hash dice "0". Almacenemos el precio de la leche en el índice 0.



Keep going, and eventually the whole array will be full of prices.

Continúe y eventualmente toda la gama estará llena de precios.

1.49	0.79	2.49	0.67	1.49
------	------	------	------	------

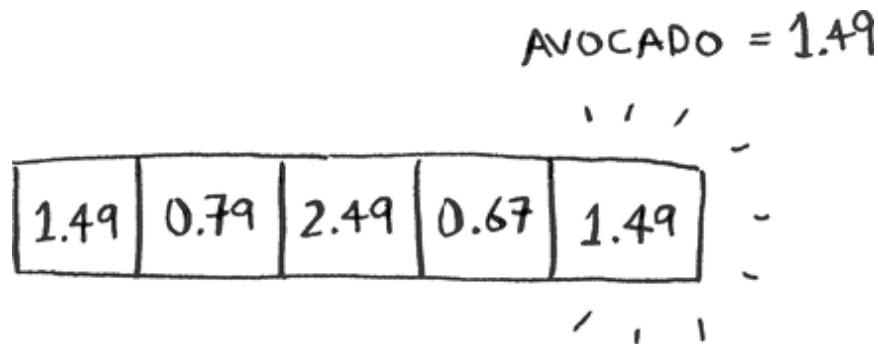
Now you ask, "Hey, what's the price of an avocado?" You don't need to search for it in the array. Just feed "avocado" into the hash function.

Ahora preguntas: "Oye, ¿cuál es el precio de un aguacate?"  
No es necesario buscarlo en la matriz. Simplemente  
introduzca "aguacate" en la función hash.



It tells you that the price is stored at index 4. And sure enough, there it is.

Te dice que el precio está almacenado en el índice 4. Y efectivamente, ahí está.



The hash function tells you exactly where the price is stored, so you don't have to search at all! This works because

La función hash le indica exactamente dónde se almacena el precio, para que no tenga que buscar en absoluto! Esto funciona porque

- The hash function consistently maps a name to the same index. Every time you put in “avocado,” you’ll get the same number back. So you can use it the first time to find where to store the price of an avocado, and then you can use it to find where you stored that price.

La función hash asigna consistentemente un nombre al mismo índice. Cada vez que ingreses "aguacate", obtendrás el mismo número. Así que puedes usarlo la primera vez para encontrar dónde almacenar el precio de un aguacate, y luego puedes usarlo para encontrar dónde almacenaste ese precio.

- The hash function maps different strings to different indexes. “Avocado” maps to index 4. “Milk” maps to index 0. Everything maps to a different slot in the array where you can store its price.

La función hash asigna diferentes cadenas a diferentes índices. “Aguacate” se asigna al índice 4. “Leche” se asigna al índice 0. Todo se asigna a una ranura diferente en la matriz donde puede almacenar su precio.

- The hash function knows how big your array is and only returns valid indexes. So if your array is five items, the hash function doesn’t return 100—that wouldn’t be a valid index in the array.

La función hash sabe qué tan grande es su matriz y solo devuelve índices válidos. Entonces, si su matriz tiene cinco elementos, la función hash no devuelve 100; ese no sería un índice válido en la matriz.

You just built a “Maggie”! Put a hash function and an array together, and you get a data structure called a *hash table*. A

hash table is the first data structure you'll learn that has some extra logic behind it. Arrays and lists map straight to memory, but hash tables are smarter. They use a hash function to intelligently figure out where to store elements.

¡Acabas de construir una "Maggie"! Junte una función hash y una matriz y obtendrá una estructura de datos llamada tabla hash. Una tabla hash es la primera estructura de datos que aprenderá y que tiene alguna lógica adicional detrás. Las matrices y listas se asignan directamente a la memoria, pero las tablas hash son más inteligentes. Utilizan una función hash para determinar de forma inteligente dónde almacenar elementos.

Hash tables are probably the most useful complex data structure you'll learn. They're also known as hash maps, maps, dictionaries, and associative arrays. And hash tables are fast! Remember our discussion of arrays and linked lists back in chapter 2? You can get an item from an array instantly. And hash tables use an array to store the data, so they're equally fast.

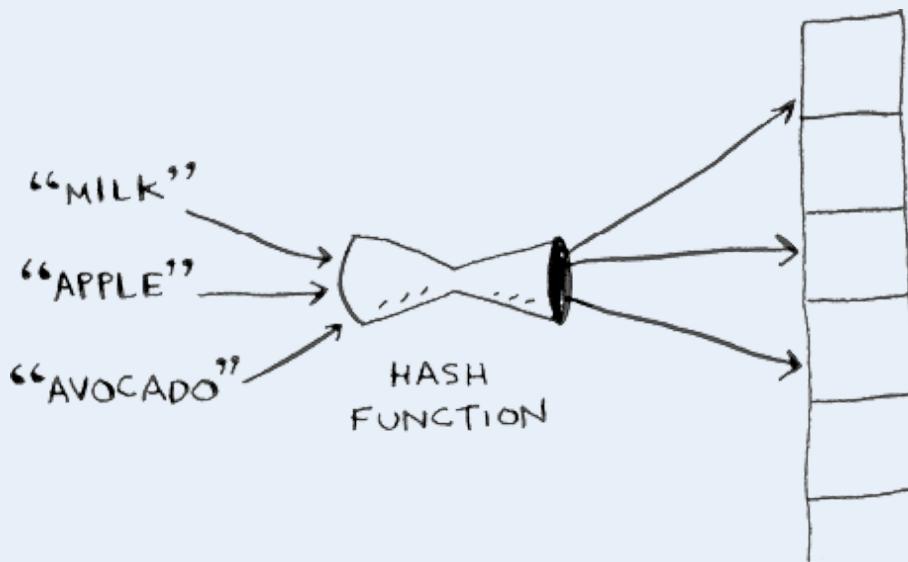
Las tablas hash son probablemente la estructura de datos compleja más útil que aprenderá. También se les conoce como mapas hash, mapas, diccionarios y matrices asociativas. ¡Y las tablas hash son rápidas! ¿Recuerda nuestra discusión sobre matrices y listas enlazadas en el capítulo 2? Puede obtener un elemento de una matriz al instante. Y las tablas hash utilizan una matriz para almacenar los datos, por lo que son igualmente rápidas.

## What's the catch?

### ¿Cuál es el truco?

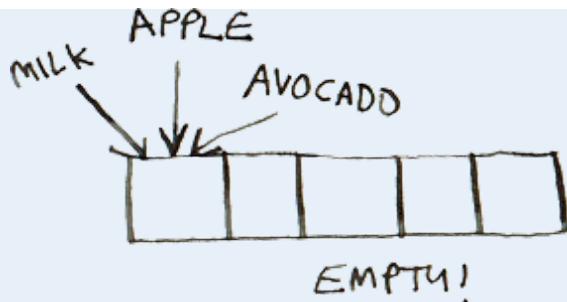
The hash function we just saw is called a *perfect hash function*. It deftly maps each grocery item to its very own slot in the array:

La función hash que acabamos de ver se llama función hash perfecta. Asigna hábilmente cada artículo comestible a su propia ranura en la matriz:



Look at them all snug in their own slots. In reality, you probably won't get a perfect one-to-one mapping like this. Your items will need to share a room. Some grocery items will map to the same slot, while other slots will go empty.

Míralos a todos cómodamente en sus propios espacios. En realidad, probablemente no obtendrá un mapeo uno a uno perfecto como este. Tus artículos deberán compartir una habitación. Algunos artículos comestibles se asignarán al mismo espacio, mientras que otros espacios quedarán vacíos.



There's a section on collisions coming up that discusses this. For now, just know that while hash tables are very useful, they rarely map items to slots so perfectly.

Próximamente habrá una sección sobre colisiones que analiza esto. Por ahora, solo debes saber que, si bien las tablas hash son muy útiles, rara vez asignan elementos a las ranuras de manera tan perfecta.

By the way, this kind of one-to-one mapping is called an *injective function*. Use that to impress your friends!

Por cierto, este tipo de mapeo uno a uno se llama función inyectiva. ¡Usa eso para impresionar a tus amigos!

You'll probably never have to implement hash tables yourself. Any good language will have an implementation for hash tables. Python has hash tables; they're called *dictionaries*. You can make a new hash table using empty braces:

Probablemente nunca tendrás que implementar tablas hash tú mismo. Cualquier buen lenguaje tendrá una implementación para tablas hash. Python tiene tablas hash; se llaman diccionarios. Puede crear una nueva tabla hash usando llaves vacías:

```
>>> book = {}
```



AN EMPTY  
HASH TABLE

*book* is a new hash table. Let's add some prices to *book*:

El libro es una nueva tabla hash. Agreguemos algunos precios a *book*:

```
>>> book["apple"] = 0.67      ①
>>> book["milk"] = 1.49       ②
>>> book["avocado"] = 1.49
>>> print(book)
{'avocado': 1.49, 'apple': 0.67, 'milk': 1.49}
```

- ① An apple costs 67 cents.
- ① Una manzana cuesta 67 céntimos.
- ② Milk costs \$1.49.
- ② La leche cuesta \$1,49.

A HASH TABLE OF  
PRODUCE PRICES

Pretty easy! Now let's ask for the price of an avocado:

Muy fácil! Ahora preguntemos el precio de un aguacate:

```
>>> print(book["avocado"])
1.49
```

① The price of an avocado

① El precio de un aguacate

A hash table has keys and values. In the `book` hash, the names of produce are the keys, and their prices are the values. A hash table maps keys to values.

Una tabla hash tiene claves y valores. En el hash `book`, los nombres de los productos son las claves y sus precios son los valores. Una tabla hash asigna claves a valores.

In the next section, you'll see some examples where hash tables are really useful.

En la siguiente sección, verá algunos ejemplos en los que las tablas hash son realmente útiles.

## EXERCISES

### EJERCICIOS

It's important for hash functions to consistently return the same output for the same input. If they don't, you won't be able to find your item after you put it in the hash table!

Es importante que las funciones hash devuelvan consistentemente el mismo resultado para la misma entrada. Si no es así, no podrás encontrar tu artículo después de colocarlo en la tabla hash!

Which of these hash functions are consistent?

¿Cuáles de estas funciones hash son consistentes?

**5.1**  $f(x) = 1$  ← Returns 1 for all input

5.1  $f(x) = 1$  ← Devuelve 1 para todas las entradas

**5.2**  $f(x) = \text{rand}()$  ← Returns a random number every time

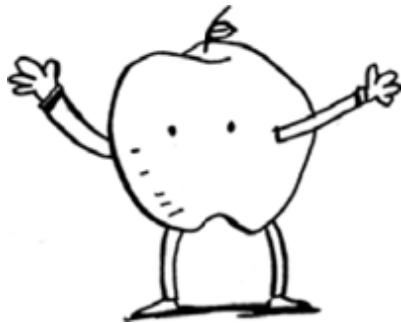
5.2  $f(x) = \text{rand}()$  ← Devuelve un número aleatorio cada vez

**5.3**  $f(x) = \text{next\_empty\_slot}()$  ← Returns the index of the next empty slot in the hash table

5.3  $f(x) = \text{next\_empty\_slot}()$  ← Devuelve el índice del siguiente espacio vacío en la tabla hash

**5.4**  $f(x) = \text{len}(x)$  ← Uses the length of the string as the index

5.4  $f(x) = \text{len}(x)$  ← Utiliza la longitud de la cadena como índice



## Use cases

### Casos de uso

Hash tables are used everywhere. This section will show you a few use cases.

Las tablas hash se utilizan en todas partes. Esta sección le mostrará algunos casos de uso.

### Using hash tables for lookups

### Usar tablas hash para búsquedas

Your phone has a handy phonebook built in. Each name has a phone number associated with it.

Su teléfono tiene una práctica agenda integrada. Cada nombre tiene un número de teléfono asociado.



BADE MAMA → 581 660 9820

ALEX MANNING → 484 234 4680

JANE MARIN → 415 567 3579

Suppose you want to build a phone book like this. You're mapping people's names to phone numbers. Your phone book needs to have this functionality:

Supongamos que desea crear una guía telefónica como esta. Estás asignando nombres de personas a números de teléfono. Su directorio telefónico debe tener esta funcionalidad:

- Add a person's name and the phone number associated with that person.

Agregue el nombre de una persona y el número de teléfono asociado con esa persona.

- Enter a person's name and get the phone number associated with that name.

Ingresé el nombre de una persona y obtenga el número de teléfono asociado con ese nombre.

This is a perfect use case for hash tables! Hash tables are great when you want to

¡Este es un caso de uso perfecto para tablas hash! Las tablas hash son geniales cuando quieras

- Create a mapping from one thing to another thing  
Crear un mapeo de una cosa a otra
- Look something up  
Busca algo

Building a phone book is pretty easy. First, make a new hash table:

Crear una guía telefónica es bastante fácil. Primero, crea una nueva tabla hash:

```
>>> phone_book = {}
```

Let's add the phone numbers of some people to this phone book:

Agreguemos los números de teléfono de algunas personas a esta guía telefónica:

```
>>> phone_book["jenny"] = "8675309"  
>>> phone_book["emergency"] = "911"
```

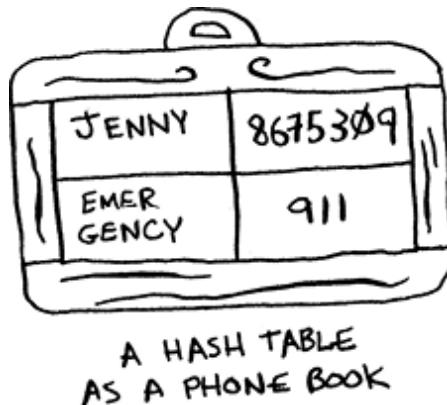
That's all there is to it! Now suppose you want to find Jenny's phone number. Just pass the key in to the hash:

¡Eso es todo al respecto! Ahora suponga que quiere encontrar el número de teléfono de Jenny. Simplemente pasa la clave al hash:

```
>>> print(phone_book["jenny"])  
8675309  
①
```

① Jenny's phone number

① Número de teléfono de Jenny



Imagine if you had to do this using an array instead. How would you do it? Hash tables make it easy to model a relationship from one item to another.

Imagínese si tuviera que hacer esto usando una matriz. ¿Como lo harias? Las tablas hash facilitan el modelado de una relación entre un elemento y otro.

Hash tables are used for lookups on a much larger scale. For example, suppose you go to a website like <http://adit.io>. Your computer has to translate adit.io to an IP address.

Las tablas hash se utilizan para búsquedas a una escala mucho mayor. Por ejemplo, supongamos que visita un sitio web como http://adit.io. Su computadora tiene que traducir adit.io a una dirección IP.

ADIT.IO → 173.255.248.55

For any website you go to, the address has to be translated to an IP address.

Para cualquier sitio web al que visite, la dirección debe traducirse a una dirección IP.

GOOGLE.COM → 74.125.239.133

FACEBOOK.COM → 173.252.128.6

SCRIBD.COM → 23.235.47.175

Wow, mapping a web address to an IP address? Sounds like a perfect use case for hash tables! This process is called *DNS resolution*. Hash tables are one way to provide this functionality. Your computer has a *DNS cache*, which stores

this mapping for websites you have recently visited, and a good way to build a DNS cache is to use a hash table.

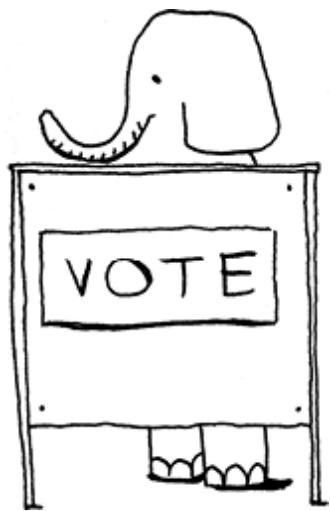
Vaya, ¿asignar una dirección web a una dirección IP? ¡Sueno como un caso de uso perfecto para tablas hash! Este proceso se llama resolución DNS. Las tablas hash son una forma de proporcionar esta funcionalidad. Su computadora tiene un caché DNS, que almacena esta asignación para los sitios web que visitó recientemente, y una buena manera de crear un caché DNS es usar una tabla hash.

## Preventing duplicate entries

### Evitar entradas duplicadas

Suppose you're running a voting booth. Naturally, every person can vote just once. How do you make sure they haven't voted before? When someone comes in to vote, you ask for their full name. Then you check it against the list of people who have voted.

Supongamos que tiene una cabina de votación. Naturalmente, cada persona puede votar sólo una vez. ¿Cómo se asegura de que no hayan votado antes? Cuando alguien viene a votar, le preguntas su nombre completo. Luego lo comparas con la lista de personas que han votado.



If their name is on the list, this person has already voted—kick them out! Otherwise, you add their name to the list and let them vote. Now suppose a lot of people have come in to vote, and the list of people who have voted is really long.

Si su nombre está en la lista, esta persona ya votó: iéchala!  
De lo contrario, agrega su nombre a la lista y les deja votar.  
Ahora supongamos que mucha gente ha venido a votar y la lista de personas que han votado es muy larga.



Each time someone new comes in to vote, you have to scan this giant list to see if they've already voted. But there's a better way: use a hash!

Cada vez que alguien nuevo llega a votar, debes escanear esta lista gigante para ver si ya votó. Pero hay una manera mejor: iusa un hash!

First, make a hash to keep track of the people who have voted:

Primero, haz un hash para realizar un seguimiento de las personas que han votado:

```
>>> voted = {}
```

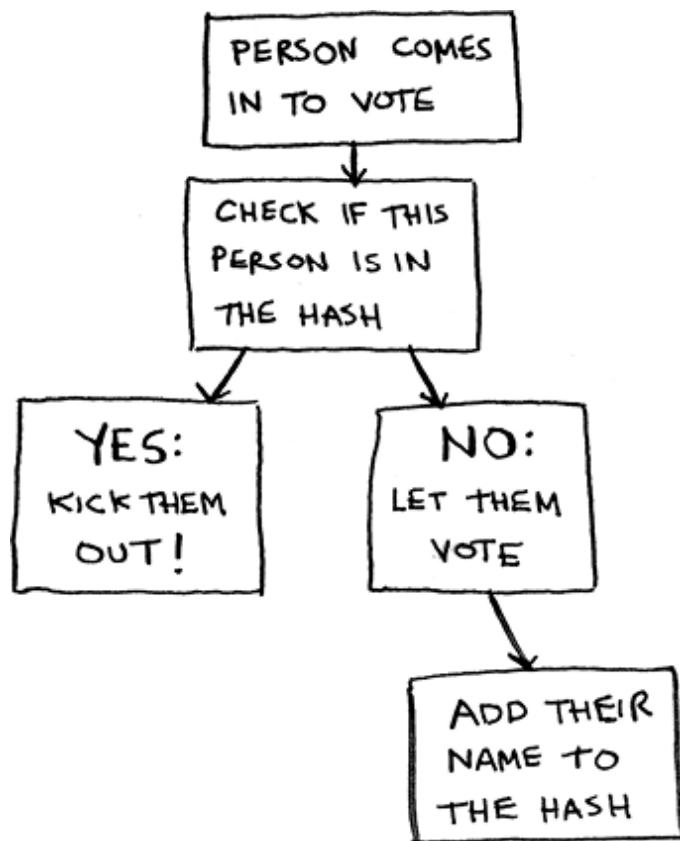
When someone new comes in to vote, check if they're already in the hash:

Cuando alguien nuevo ingrese a votar, verifique si ya está en el hash:

```
>>> value = "tom" in voted
```

*value* is now `True` if "tom" is in the hash table. Otherwise, it is `False`. You can use this to check whether someone has already voted!

El valor ahora es `True` si "tom" está en la tabla hash. De lo contrario, es `False`. ¡Puedes utilizar esto para comprobar si alguien ya ha votado!



Here's the code:

Aquí está el código:

```
voted = {}  
def check_voter(name):  
    if name in voted:  
        print("kick them out!")  
    else:  
        voted[name] = True  
        print("let them vote!")
```

Let's test it a few times:

Probémoslo varias veces:

```
>>> check_voter("tom")  
let them vote!  
>>> check_voter("mike")  
let them vote!  
>>> check_voter("mike")  
kick them out!
```

The first time Tom goes in, the hash table will print, "let them vote!" Then Mike goes in, and it prints, "let them vote!" Then Mike tries to go a second time, and it prints, "kick them out!"

La primera vez que Tom ingresa, la tabla hash imprimirá: "iDéjenlos votar!" Luego Mike entra y escribe: "iDéjenlos votar!" Luego Mike intenta ir por segunda vez y dice "iéchalo!".

Remember, if you were storing these names in a list of people who have voted, this function would eventually become really slow, because it would have to run a simple search over the entire list. But you're storing their names in a hash table instead, and a hash table instantly tells you whether this person's name is in the hash table or not. Checking for duplicates is very fast with a hash table.

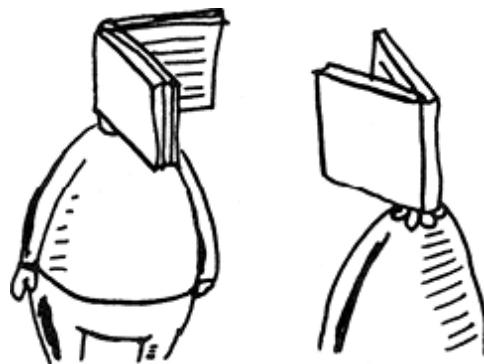
Recuerde, si estuviera almacenando estos nombres en una lista de personas que han votado, esta función eventualmente se volvería muy lenta, porque tendría que ejecutar una búsqueda simple en toda la lista. Pero en su lugar, estás almacenando sus nombres en una tabla hash, y una tabla hash te dice instantáneamente si el nombre de esta persona está en la tabla hash o no. La búsqueda de duplicados es muy rápida con una tabla hash.

## Using hash tables as a cache

### Usar tablas hash como caché

One final use case: caching. If you work on a website, you may have heard of caching before as a good thing to do. Here's the idea. Suppose you visit <https://facebook.com>:

Un último caso de uso: el almacenamiento en caché. Si trabaja en un sitio web, es posible que haya oido hablar antes del almacenamiento en caché como algo bueno. Aquí está la idea. Supongamos que visita <https://facebook.com>:



1. You make a request to Facebook's server.

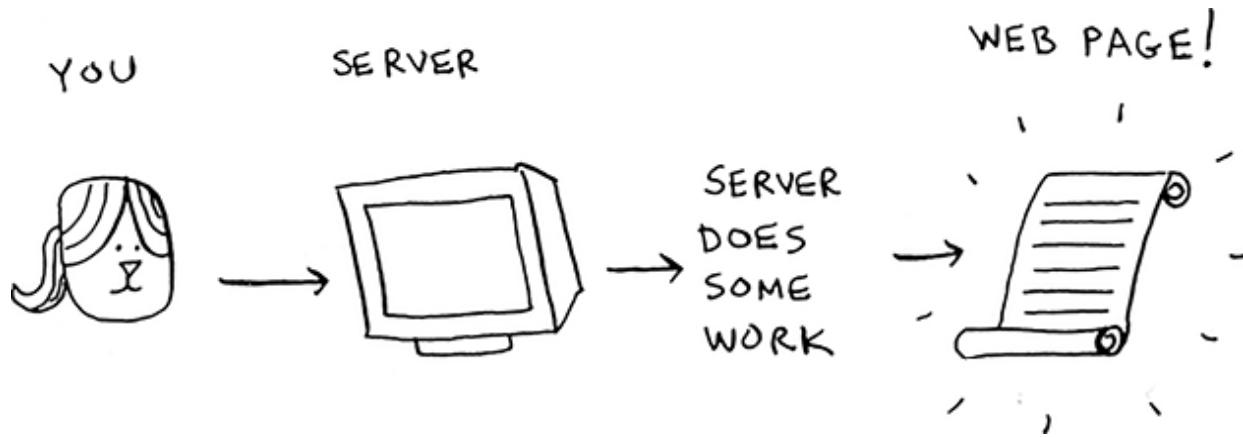
Realizas una solicitud al servidor de Facebook.

2. The server thinks for a second and comes up with the web page to send you.

El servidor piensa por un segundo y muestra la página web para enviarte.

3. You get a web page.

Obtienes una página web.



For example, on Facebook, the server may be collecting all of your friends' activity to show you. It takes a couple of seconds to collect all that activity and show it to you. As a user, that couple of seconds can feel like a long time. You

might think, "Why is Facebook being so slow?" On the other hand, Facebook's servers have to serve millions of people, and that couple of seconds adds up. Facebook's servers are really working hard to serve all of those websites. Is there a way to have Facebook's servers do less work?

Por ejemplo, en Facebook, el servidor puede estar recopilando toda la actividad de tus amigos para mostrártela. Se necesitan un par de segundos para recopilar toda esa actividad y mostrártela. Como usuario, ese par de segundos puede parecer mucho tiempo. Podrías pensar: "¿Por qué Facebook está siendo tan lento?" Por otro lado, los servidores de Facebook tienen que servir a millones de personas, y ese par de segundos suma. Los servidores de Facebook están trabajando muy duro para servir a todos esos sitios web. ¿Existe alguna manera de hacer que los servidores de Facebook trabajen menos?

Suppose you have a niece who keeps asking you about planets. "How far is Mars from Earth?" "How far is the Moon?" "How far is Jupiter?" Each time, you have to do a Google search and give her an answer. It takes a couple of minutes. Now, suppose she always asked, "How far is the Moon?" Pretty soon, you'd memorize that the Moon is 238,900 miles away. You wouldn't have to look it up on Google; you'd just remember and answer. This is how caching works: websites remember the data instead of recalculating it.

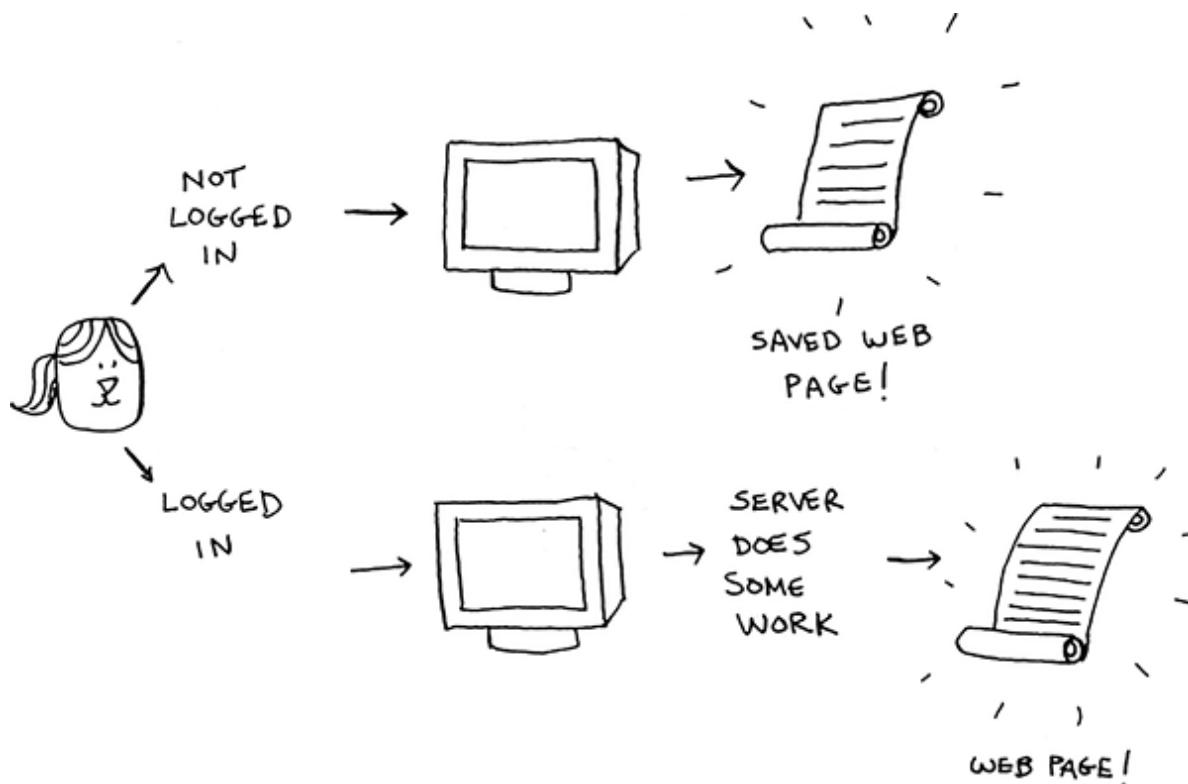
Supongamos que tienes una sobrina que sigue preguntándote sobre los planetas. "¿A qué distancia está Marte de la Tierra?" "¿A qué distancia está la Luna?" "¿A qué

distancia está Júpiter?" Cada vez, debes hacer una búsqueda en Google y darle una respuesta. Tarda un par de minutos. Ahora bien, supongamos que ella siempre preguntara: "¿A qué distancia está la Luna?" Muy pronto memorizarías que la Luna está a 238.900 millas de distancia. No tendrías que buscarlo en Google; simplemente lo recordarías y responderías. Así es como funciona el almacenamiento en caché: los sitios web recuerdan los datos en lugar de volver a calcularlos.

If you're logged in to Facebook, all the content you see is tailored just for you. Each time you go to <https://facebook.com>, its servers have to think about what content you're interested in. But if you're not logged in to Facebook, you see the login page. Everyone sees the same login page. Facebook is asked the same thing over and over: "Give me the home page when I'm logged out." So it stops making the server do work to figure out what the home page looks like. Instead, it memorizes what the home page looks like and sends it to you.

Si ha iniciado sesión en Facebook, todo el contenido que ve está diseñado exclusivamente para usted. Cada vez que visita <https://facebook.com>, sus servidores tienen que pensar en el contenido que le interesa. Pero si no ha iniciado sesión en Facebook, verá la página de inicio de sesión. Todos ven la misma página de inicio de sesión. A Facebook se le pregunta lo mismo una y otra vez: "Dame la página de inicio cuando cierre la sesión". Por lo tanto, deja de hacer que el servidor funcione para determinar cómo se ve la página de

inicio. En cambio, memoriza cómo se ve la página de inicio y te la envía.



This is called *caching*. It has two advantages:

Esto se llama almacenamiento en caché. Tiene dos ventajas:

- You get the web page a lot faster, just like when you memorized the distance from Earth to the Moon. The next time your niece asks you, you won't have to Google it. You can answer instantly.

Obtienes la página web mucho más rápido, como cuando memorizaste la distancia de la Tierra a la Luna. La próxima vez que tu sobrina te pregunte, no tendrás que buscarlo en Google. Puedes responder al instante.

- Facebook has to do less work.

Facebook tiene que hacer menos trabajo.

Caching is a common way to make things faster. All big websites use caching. And that data is cached in a hash!

El almacenamiento en caché es una forma común de acelerar las cosas. Todos los sitios web grandes utilizan el almacenamiento en caché. ¡Y esos datos se almacenan en caché en un hash!

Facebook isn't just caching the home page. It's also caching the About page, the Contact page, the Terms and Conditions page, and a lot more. So it needs a mapping from page URL to page data.

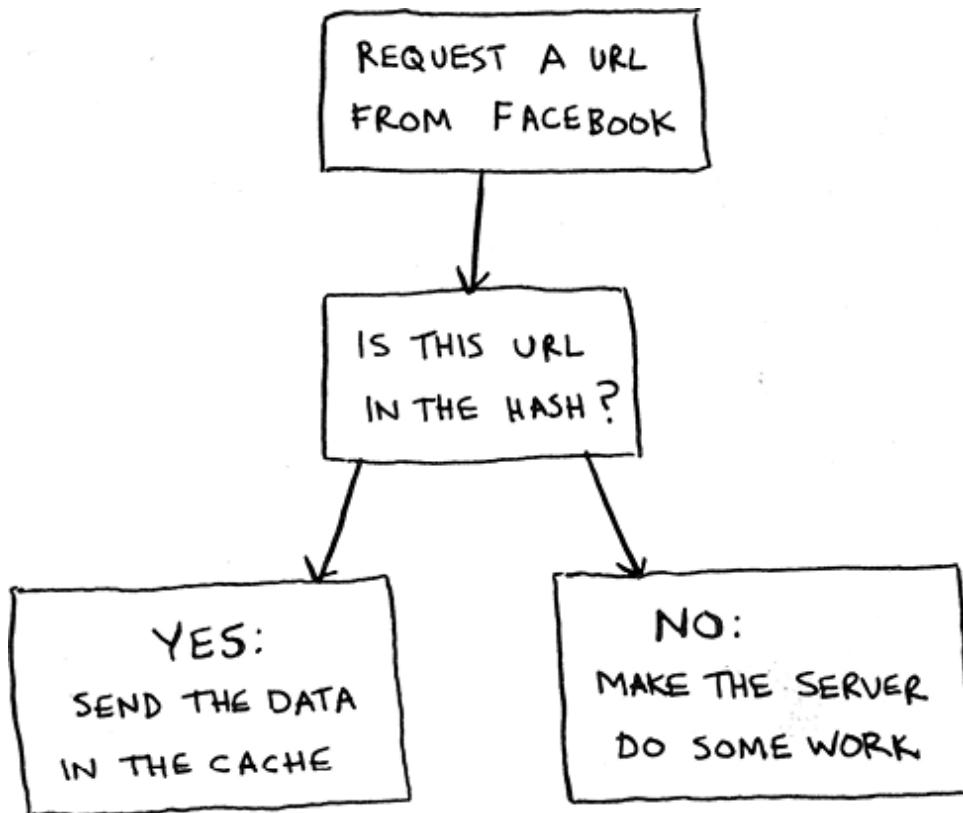
Facebook no solo almacena en caché la página de inicio. También almacena en caché la página Acerca de, la página Contacto, la página Términos y condiciones y mucho más. Por lo tanto, necesita una asignación de la URL de la página a los datos de la página.

*facebook.com/about → DATA FOR THE ABOUT PAGE*

*facebook.com → DATA FOR THE HOME PAGE*

When you visit a page on Facebook, it first checks whether the page is stored in the hash.

Cuando visita una página en Facebook, primero verifica si la página está almacenada en el hash.



Here it is in pseudocode:

Aquí está en pseudocódigo:

```

cache = {}

def get_page(url):
    if url in cache:
        return cache[url]      ①
    else:
        data = get_data_from_server(url)
        cache[url] = data      ②
        return data
  
```

① Returns cached data

① Devuelve datos almacenados en caché

② Saves this data in your cache first

② Primero guarda estos datos en su caché

Here, you make the server do work only if the URL isn't in the cache. Before you return the data, though, you save it in the cache. The next time someone requests this URL, you can send the data from the cache instead of making the server do the work.

Aquí, haces que el servidor funcione sólo si la URL no está en el caché. Sin embargo, antes de devolver los datos, los guarda en el caché. La próxima vez que alguien solicite esta URL, podrá enviar los datos desde la caché en lugar de dejar que el servidor haga el trabajo.

## Recap

## Resumen

To recap, hashes are good for

En resumen, los hashes son buenos para

- Modeling relationships from one thing to another thing  
Modelar relaciones de una cosa a otra
- Filtering out duplicates  
Filtrar duplicados
- Caching/memoizing data instead of making your server do work  
Almacenar en caché/memorizar datos en lugar de hacer que su servidor funcione

## **Collisions**

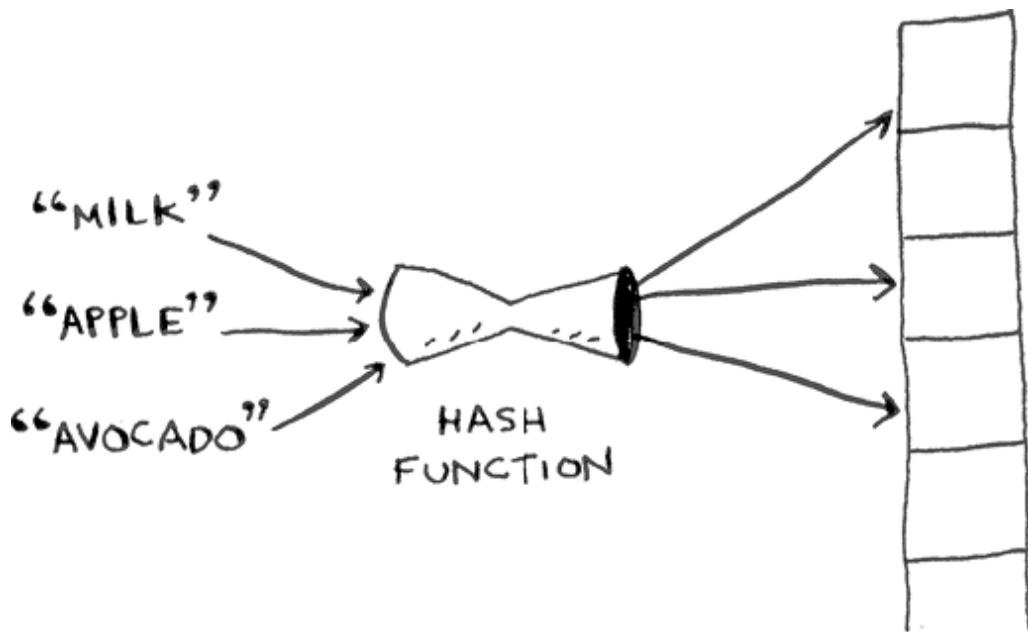
### **Colisiones**

Like I said earlier, most languages have hash tables. You don't need to know how to write your own. So, I won't talk about the internals of hash tables too much. But you still care about performance! To understand the performance of hash tables, you first need to understand what collisions are. The next two sections cover collisions and performance.

Como dije antes, la mayoría de los idiomas tienen tablas hash. No es necesario que sepas escribir el tuyo propio. Por lo tanto, no hablaré demasiado sobre los aspectos internos de las tablas hash. ¡Pero todavía te importa el rendimiento! Para comprender el rendimiento de las tablas hash, primero debe comprender qué son las colisiones. Las dos secciones siguientes cubren colisiones y rendimiento.

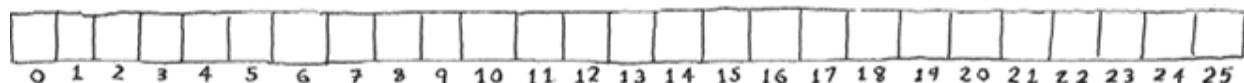
First, I've been telling you a white lie. I told you that a hash function always maps different keys to different slots in the array.

Primero, te he estado diciendo una mentira piadosa. Te dije que una función hash siempre asigna diferentes claves a diferentes ranuras en la matriz.



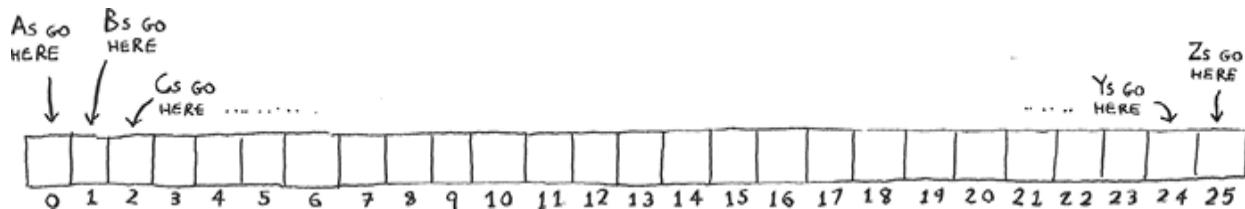
In reality, it's almost impossible to write a hash function that does this. Let's take a simple example. Suppose your array contains 26 slots.

En realidad, es casi imposible escribir una función hash que haga esto. Tomemos un ejemplo sencillo. Supongamos que su matriz contiene 26 ranuras.



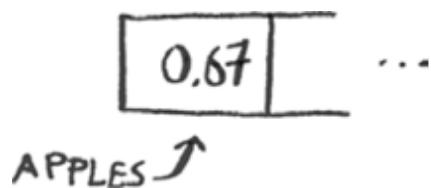
And your hash function is really simple: it assigns a spot in the array alphabetically.

Y su función hash es realmente simple: asigna alfabéticamente un lugar en la matriz.



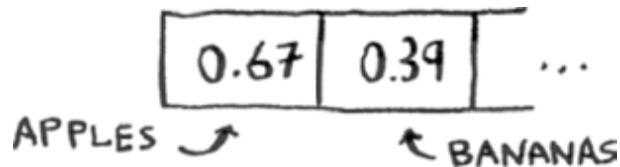
Maybe you can already see the problem. You want to put the price of apples in your hash. You get assigned the first slot.

Quizás ya puedes ver el problema. Quieres poner el precio de las manzanas en tu hash. Te asignan el primer puesto.



Then you want to put the price of bananas in the hash. You get assigned the second slot.

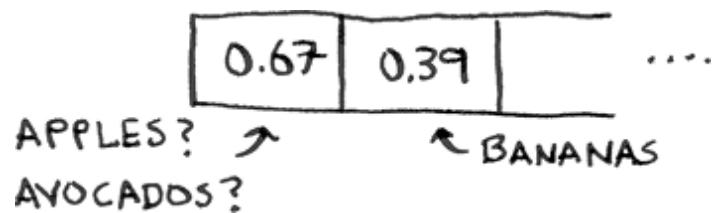
Entonces quieres poner el precio de los plátanos en el hachís. Se te asigna el segundo puesto.



Everything is going so well! But now you want to put the price of avocados in your hash. You get assigned the first slot again.

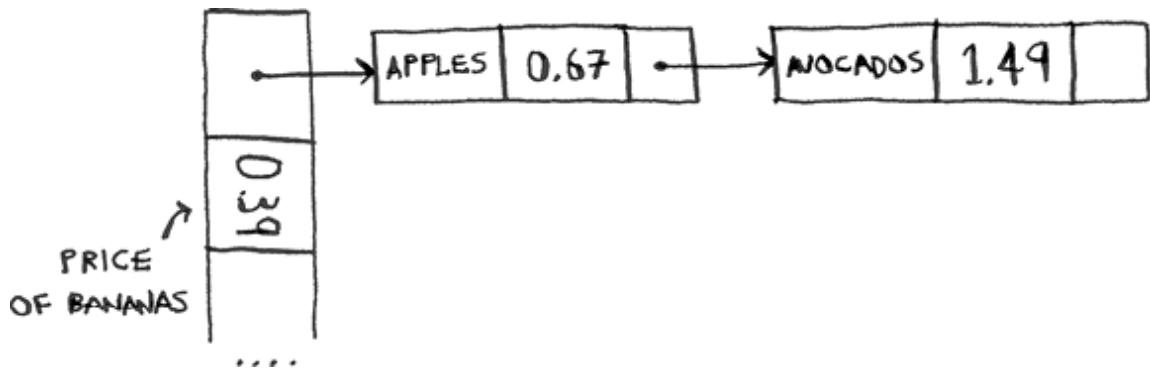
¡Todo va tan bien! Pero ahora quieres poner el precio de los aguacates en tu hachís. Se te asigna nuevamente el primer

espacio.



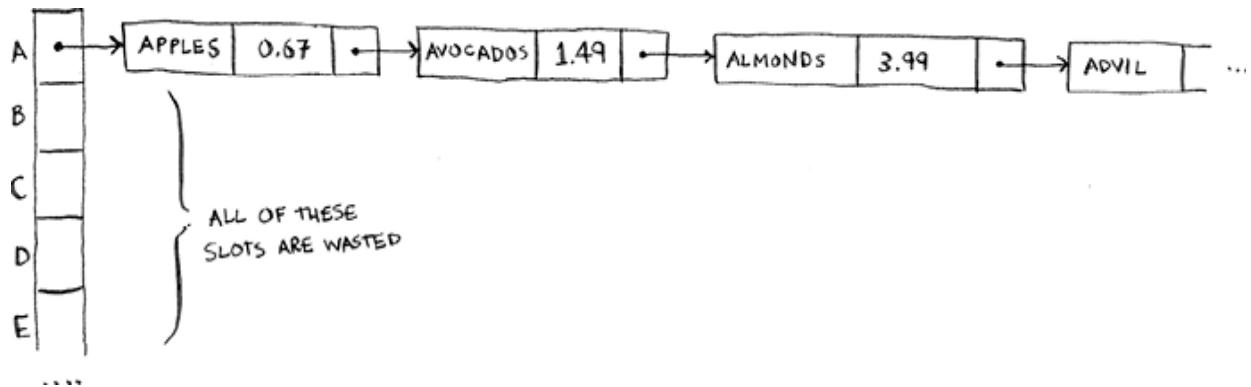
Oh no! Apples have that slot already! What to do? This is called a *collision*: two keys have been assigned the same slot. This is a problem. If you store the price of avocados at that slot, you'll overwrite the price of apples. Then the next time someone asks for the price of apples, they will get the price of avocados instead! Collisions are bad, and you need to work around them. There are many different ways to deal with collisions. The simplest one is this: if multiple keys map to the same slot, start a linked list at that slot.

¡Oh, no! ¡Las manzanas ya tienen ese espacio! ¿Qué hacer? Esto se llama colisión: a dos claves se les ha asignado la misma ranura. Esto es un problema. Si almacena el precio de los aguacates en ese espacio, sobrescribirá el precio de las manzanas. Entonces, la próxima vez que alguien pregunte el precio de las manzanas, obtendrá el precio de los aguacates! Las colisiones son malas y es necesario solucionarlas. Hay muchas maneras diferentes de afrontar las colisiones. La más simple es esta: si varias claves se asignan a la misma ranura, inicie una lista vinculada en esa ranura.



In this example, both “apples” and “avocados” map to the same slot. So you start a linked list at that slot. If you need to know the price of bananas, it’s still quick. If you need to know the price of apples, it’s a little slower. You have to search through this linked list to find “apples.” If the linked list is small, no big deal—you have to search through three or four elements. But suppose you work at a grocery store where you only sell produce that starts with the letter A.

En este ejemplo, tanto “manzanas” como “aguacates” se asignan al mismo espacio. Entonces comienzas una lista vinculada en ese espacio. Si necesitas saber el precio de los plátanos, todavía es rápido. Si necesitas saber el precio de las manzanas, es un poco más lento. Tienes que buscar en esta lista vinculada para encontrar “manzanas”. Si la lista enlazada es pequeña, no es gran cosa: debe buscar entre tres o cuatro elementos. Pero supongamos que trabaja en una tienda de comestibles donde solo vende productos que comienzan con la letra A.



Hey, wait a minute! The entire hash table is totally empty except for one slot. And that slot has a giant linked list! Every single element in this hash table is in the linked list. That's as bad as putting everything in a linked list to begin with. It's going to slow down your hash table.

Oye, espera un minuto! Toda la tabla hash está totalmente vacía excepto una ranura. ¡Y esa tragamonedas tiene una lista de enlaces gigante! Cada elemento de esta tabla hash está en la lista vinculada. Para empezar, eso es tan malo como poner todo en una lista vinculada. Va a ralentizar tu tabla hash.

There are two lessons here:

Hay dos lecciones aquí:

- *Your hash function is really important.* Your hash function mapped all the keys to a single slot. Ideally, your hash function would map keys evenly all over the hash.

Tu función hash es realmente importante. Su función hash asignó todas las claves a una sola ranura.

Idealmente, su función hash asignaría claves de manera uniforme en todo el hash.

- If those linked lists get long, it slows down your hash table a lot. But they won't get long if you *use a good hash function!*

Si esas listas enlazadas se alargan, ralentiza mucho la tabla hash. ¡Pero no tardarán mucho si utilizas una buena función hash!

Hash functions are important. A good hash function will give you very few collisions. So how do you pick a good hash function? That's coming up in the next section!

Las funciones hash son importantes. Una buena función hash generará muy pocas colisiones. Entonces, ¿cómo se elige una buena función hash? ¡Eso viene en la siguiente sección!

## Performance

## Actuación

You started this chapter at the grocery store. You wanted to build something that would give you the prices for produce

*instantly.* Well, hash tables are really fast.

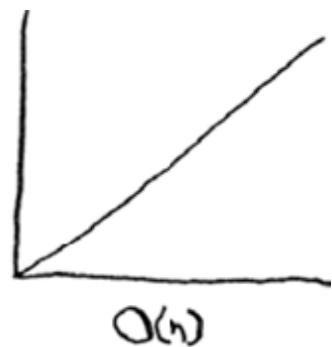
Comenzaste este capítulo en el supermercado. Querías crear algo que te proporcionara los precios de los productos al instante. Bueno, las tablas hash son realmente rápidas.

	AVERAGE CASE	WORST CASE
SEARCH	$O(1)$	$O(n)$
INSERT	$O(1)$	$O(n)$
DELETE	$O(1)$	$O(n)$

PERFORMANCE OF HASH TABLES

In the average case, hash tables take  $O(1)$  for everything.  $O(1)$  is called *constant time*. You haven't seen constant time before. It doesn't mean instant. It means the time taken will stay the same, regardless of how big the hash table is. For example, you know that simple search takes linear time.

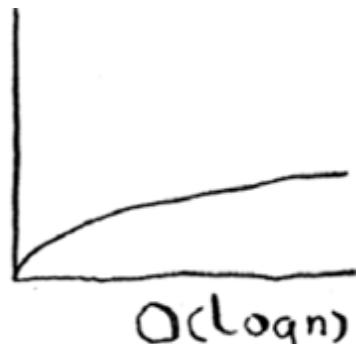
En el caso promedio, las tablas hash toman  $O(1)$  para todo.  $O(1)$  se llama tiempo constante. No has visto un tiempo constante antes. No significa instantáneo. Significa que el tiempo necesario seguirá siendo el mismo, independientemente del tamaño de la tabla hash. Por ejemplo, sabes que una búsqueda simple requiere un tiempo lineal.



$O(n)$   
LINEAR TIME  
(SIMPLE SEARCH)

Binary search is faster—it takes log time.

La búsqueda binaria es más rápida: requiere tiempo de registro.



$O(\log n)$   
LOG TIME  
(BINARY SEARCH)

Looking something up in a hash table takes constant time.

Buscar algo en una tabla hash lleva un tiempo constante.



See how it's a flat line? That means it doesn't matter whether your hash table has 1 element or 1 billion elements —getting something out of a hash table will take the same amount of time. Actually, you've seen constant time before. Getting an item out of an array takes constant time. It doesn't matter how big your array is; it takes the same amount of time to get an element. In the average case, hash tables are really fast.

¿Ves cómo es una línea plana? Eso significa que no importa si su tabla hash tiene 1 elemento o mil millones de elementos: obtener algo de una tabla hash llevará la misma cantidad de tiempo. En realidad, has visto tiempo constante antes. Sacar un elemento de una matriz lleva un tiempo constante. No importa qué tan grande sea su conjunto; se necesita la misma cantidad de tiempo para obtener un elemento. En el caso normal, las tablas hash son realmente rápidas.

In the worst case, a hash table takes  $O(n)$ —linear time—for everything, which is really slow. Let's compare hash tables to arrays and lists.

En el peor de los casos, una tabla hash requiere  $O(n)$  (tiempo lineal) para todo, lo cual es realmente lento. Comparemos tablas hash con matrices y listas.

	HASH TABLES (AVERAGE)	HASH TABLES (WORST)	LINKED ARRAYS	LINKED LISTS
SEARCH	$O(1)$	$O(n)$	$O(1)$	$O(n)$
INSERT	$O(1)$	$O(n)$	$O(n)$	$O(1)$
DELETE	$O(1)$	$O(n)$	$O(n)$	$O(1)$

Look at the average case for hash tables. Hash tables are as fast as arrays at searching (getting a value at an index). And they're as fast as linked lists at inserts and deletes. It's the best of both worlds! But in the worst case, hash tables are slow at all of those. So it's important that you don't hit worst-case performance with hash tables. And to do that, you need to avoid collisions. To avoid collisions, you need

Mire el caso promedio de las tablas hash. Las tablas hash son tan rápidas como las matrices en la búsqueda (obtener un valor en un índice). Y son tan rápidos como listas vinculadas al insertar y eliminar. ¡Es lo mejor de ambos mundos! Pero en el peor de los casos, las tablas hash son lentas en todos ellos. Por lo tanto, es importante que no alcance el peor rendimiento con las tablas hash. Y para ello

es necesario evitar colisiones. Para evitar colisiones, es necesario

- A low load factor  
Un factor de carga bajo
- A good hash function  
Una buena función hash

**NOTE** Before you start this next section, know that it isn't required reading. I'm going to talk about how to implement a hash table, but you'll never have to do that yourself. Whatever programming language you use will have an implementation of hash tables built in. You can use the built-in hash table and assume it will have good performance. The next section gives you a peek under the hood.

Nota Antes de comenzar la siguiente sección, sepá que no es necesario leerla. Voy a hablar sobre cómo implementar una tabla hash, pero nunca tendrás que hacerlo tú mismo. Cualquiera que sea el lenguaje de programación que utilice, tendrá una implementación de tablas hash integradas. Puede utilizar la tabla hash integrada y asumir que tendrá un buen rendimiento. La siguiente sección le da un vistazo debajo del capó.

## Load factor

### Factor de carga

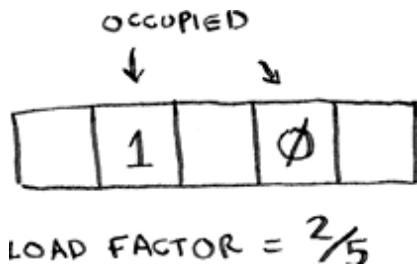
The load factor of a hash table is easy to calculate.

El factor de carga de una tabla hash es fácil de calcular.

$$\frac{\text{NUMBER OF ITEMS IN HASH TABLE}}{\text{TOTAL NUMBER OF SLOTS}}$$

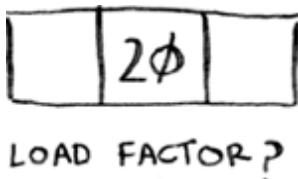
Hash tables use an array for storage, so you count the number of occupied slots in an array. For example, this hash table has a load factor of 2/5, or 0.4.

Las tablas hash utilizan una matriz para el almacenamiento, por lo que se cuenta el número de espacios ocupados en una matriz. Por ejemplo, esta tabla hash tiene un factor de carga de 2/5 o 0,4.



What's the load factor of this hash table?

¿Cuál es el factor de carga de esta tabla hash?

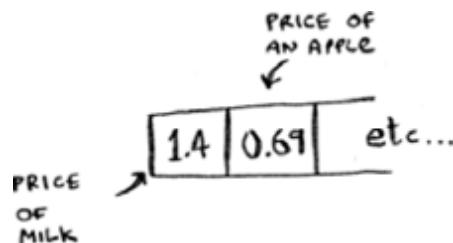


If you said 1/3, you're right. Load factor measures how full your hash table is.

Si dijiste 1/3, tienes razón. El factor de carga mide qué tan llena está su tabla hash.

Suppose you need to store the price of 100 produce items in your hash table, and your hash table has 100 slots. In the best case, each item will get its own slot.

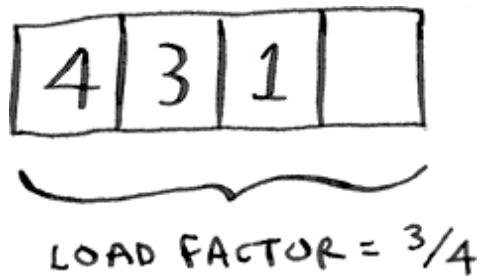
Supongamos que necesita almacenar el precio de 100 productos agrícolas en su tabla hash, y su tabla hash tiene 100 espacios. En el mejor de los casos, cada artículo tendrá su propio espacio.



This hash table has a load factor of 1. What if your hash table has only 50 slots? Then it has a load factor of 2. There's no way each item will get its own slot because there aren't enough slots! Having a load factor greater than 1 means you have more items than slots in your array. Once the load factor starts to grow, you need to add more slots to your hash table. This is called *resizing*. For example, suppose you have this hash table that is getting pretty full.

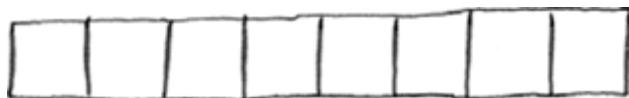
Esta tabla hash tiene un factor de carga de 1. ¿Qué pasa si su tabla hash tiene solo 50 espacios? Entonces tiene un factor de carga de 2. ¡No hay forma de que cada artículo tenga su propio espacio porque no hay suficientes espacios! Tener un factor de carga mayor que 1 significa que tienes

más elementos que espacios en tu conjunto. Una vez que el factor de carga comience a crecer, deberá agregar más espacios a su tabla hash. Esto se llama cambiar el tamaño. Por ejemplo, supongamos que tiene esta tabla hash que se está llenando bastante.



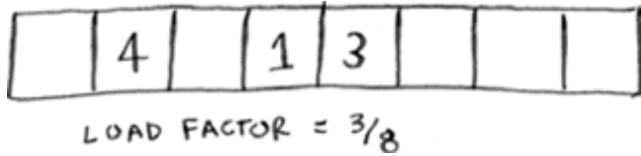
You need to resize this hash table. First, you create a new array that's bigger. The rule of thumb is to make an array twice the size of the original.

Necesita cambiar el tamaño de esta tabla hash. Primero, crea una nueva matriz que es más grande. La regla general es hacer una matriz del doble del tamaño del original.



Now you need to reinsert all of those items into this new hash table using the `hash` function.

Ahora necesita volver a insertar todos esos elementos en esta nueva tabla hash usando la función `hash`.



This new table has a load factor of  $3/8$ . Much better! With a lower load factor, you'll have fewer collisions, and your table will perform better. A good rule of thumb is to resize when your load factor is greater than 0.7.

Esta nueva mesa tiene un factor de carga de  $3/8$ . ¡Mucho mejor! Con un factor de carga más bajo, tendrá menos colisiones y su mesa funcionará mejor. Una buena regla general es cambiar el tamaño cuando el factor de carga sea superior a 0,7.

You might be thinking, "This resizing business takes a lot of time!" And you're right. Resizing is expensive, and you don't want to resize too often. But averaged out, hash tables take  $O(1)$  even with resizing.

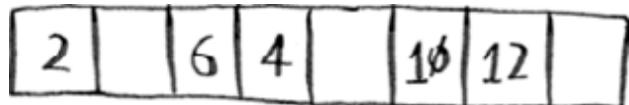
Quizás esté pensando: "¡Este negocio de cambio de tamaño requiere mucho tiempo!". Y tienes razón. Cambiar el tamaño es costoso y no conviene cambiar el tamaño con demasiada frecuencia. Pero en promedio, las tablas hash toman  $O(1)$  incluso con el cambio de tamaño.

## A good hash function

### Una buena función hash

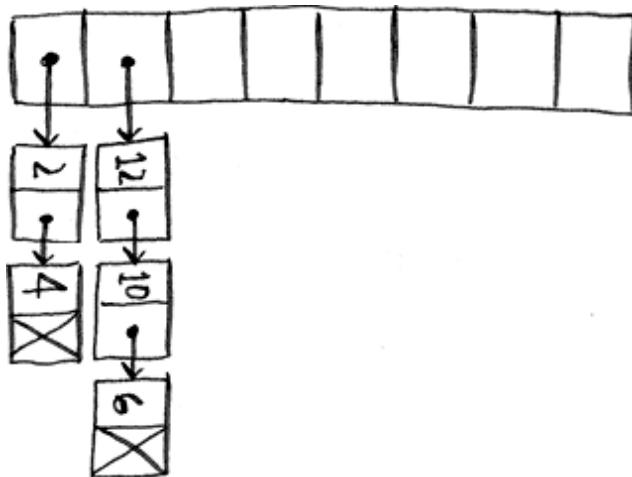
A good hash function distributes values in the array evenly.

Una buena función hash distribuye los valores de la matriz de manera uniforme.



A bad hash function groups values together and produces a lot of collisions.

Una función hash incorrecta agrupa valores y produce muchas colisiones.



What is a good hash function? That's something you'll never have to worry about—smart folks sit in dark rooms and worry about that. If you're really curious, look up CityHash. That's what Google's Abseil library uses. Abseil is an open source C++ library based on internal Google code. It provides all kinds of general-purpose C++ functions. Abseil is a building block for Google's code, so if it uses CityHash, you can be sure that CityHash is pretty good. You could use that as your hash function.

¿Qué es una buena función hash? Eso es algo de lo que nunca tendrás que preocuparte: la gente inteligente se sienta en cuartos oscuros y se preocupa por eso. Si tienes mucha curiosidad, busca CityHash. Eso es lo que utiliza la biblioteca Abseil de Google. Abseil es una biblioteca C++ de código abierto basada en el código interno de Google.

Proporciona todo tipo de funciones C++ de uso general. Abseil es un componente básico del código de Google, por lo que si utiliza CityHash, puede estar seguro de que CityHash es bastante bueno. Podrías usar eso como tu función hash.

## EXERCISES

### EJERCICIOS

It's important for hash functions to have a good distribution. They should map items as broadly as possible. The worst case is a hash function that maps all items to the same slot in the hash table.

Es importante que las funciones hash tengan una buena distribución. Deberían mapear los elementos lo más ampliamente posible. El peor de los casos es una función hash que asigna todos los elementos al mismo espacio en la tabla hash.

Suppose you have these four hash functions that work with strings:

Suponga que tiene estas cuatro funciones hash que funcionan con cadenas:

1. Return "1" for all input.  
Devuelve "1" para todas las entradas.
2. Use the length of the string as the index.  
Utilice la longitud de la cadena como índice.
3. Use the first character of the string as the index. So, all strings starting with *a* are hashed together, and so on.  
Utilice el primer carácter de la cadena como índice.  
Entonces, todas las cadenas que comienzan con *a* se combinan, y así sucesivamente.
4. Map every letter to a prime number:  $a = 2$ ,  $b = 3$ ,  $c = 5$ ,  $d = 7$ ,  $e = 11$ , and so on. For a string, the hash function is the sum of all the characters modulo the size of the hash.  
For example, if your hash size is 10, and the string is "bag," the index is  $(3 + 2 + 17) \% 10 = 22 \% 10 = 2$ .  
Asigne cada letra a un número primo:  $a = 2$ ,  $b = 3$ ,  $c = 5$ ,  $d = 7$ ,  $e = 11$ , etc. Para una cadena, la función hash es la suma de todos los caracteres módulo del tamaño del hash. Por ejemplo, si el tamaño de su hash es 10 y la cadena es "bolsa", el índice es  $(3 + 2 + 17) \% 10 = 22 \% 10 = 2$ .

For each of these examples, which hash functions would provide a good distribution? Assume a hash table size of 10 slots.

Para cada uno de estos ejemplos, ¿qué funciones hash proporcionarían una buena distribución? Supongamos un tamaño de tabla hash de 10 espacios.

**5.5** A phonebook where the keys are names and values are phone numbers. The names are as follows: Esther, Ben, Bob, and Dan.

5.5 Una agenda donde las claves son nombres y los valores son números de teléfono. Los nombres son los siguientes: Esther, Ben, Bob y Dan.

**5.6** A mapping from battery size to power. The sizes are A, AA, AAA, and AAAA.

5.6 A mapeo del tamaño de la batería a la potencia. Los tamaños son A, AA, AAA y AAAA.

**5.7** A mapping from book titles to authors. The titles are *Maus*, *Fun Home*, and *Watchmen*.

5.7 Un mapeo desde los títulos de los libros hasta los autores. Los títulos son *Maus*, *Fun Home* y *Watchmen*.

## **Recap**

## **Resumen**

- You'll almost never have to implement a hash table yourself. The programming language you use should provide an implementation for you. You can use Python's hash tables and assume that you'll get the average-case performance: constant time.

Casi nunca tendrás que implementar una tabla hash tú mismo. El lenguaje de programación que utilice debe proporcionarle una implementación. Puede utilizar las tablas hash de Python y asumir que obtendrá el rendimiento promedio: tiempo constante.

- Hash tables are a powerful data structure because they're so fast and they let you model data in a different way. You might soon find that you're using them all the time.

Las tablas hash son una estructura de datos poderosa porque son muy rápidas y te permiten modelar datos de una manera diferente. Es posible que pronto descubras que los estás usando todo el tiempo.

- You can make a hash table by combining a hash function with an array.

Puede crear una tabla hash combinando una función hash con una matriz.

- Collisions are bad. You need a hash function that minimizes collisions.  
Las colisiones son malas. Necesita una función hash que minimice las colisiones.
- Hash tables have really fast search, insert, and delete.  
Las tablas hash tienen búsqueda, inserción y eliminación realmente rápidas.
- Hash tables are good for modeling relationships from one item to another item.  
Las tablas hash son buenas para modelar relaciones de un elemento a otro.
- Once your load factor is greater than 0.7, it's time to resize your hash table.  
Una vez que su factor de carga sea mayor que 0,7, es hora de cambiar el tamaño de su tabla hash.
- Hash tables are used for caching data (for example, with a web server).  
Las tablas hash se utilizan para almacenar datos en caché (por ejemplo, con un servidor web).
- Hash tables are great for catching duplicates.  
Las tablas hash son excelentes para detectar duplicados.

# **6 Breadth-first search**

---

## **6 Búsqueda prioritaria en amplitud**

---

### **In this chapter**

#### **En este capítulo**

- You learn how to model a network using a new, abstract data structure: graphs.  
Aprenderá a modelar una red utilizando una estructura de datos nueva y abstracta: gráficos.
- You learn breadth-first search, an algorithm you can run on graphs to answer questions like, “What’s the shortest path to go to X?”  
Aprende búsqueda en amplitud, un algoritmo que puede ejecutar en gráficos para responder preguntas como "¿Cuál es el camino más corto para ir a X?"
- You learn about directed versus undirected graphs.  
Aprenderá sobre gráficos dirigidos y no dirigidos.
- You learn topological sort, a different kind of sorting algorithm that exposes dependencies between nodes.  
Aprenderá clasificación topológica, un tipo diferente de algoritmo de clasificación que expone las dependencias entre nodos.

This chapter introduces graphs. First, I'll talk about what graphs are (they don't involve an X or Y axis). Then I'll show you your first graph algorithm. It's called *breadth-first search* (BFS).

Este capítulo presenta gráficos. Primero, hablaré sobre qué son las gráficas (no involucran un eje X o Y). Luego te mostraré tu primer algoritmo gráfico. Se llama búsqueda en amplitud (BFS).

Breadth-first search allows you to find the shortest distance between two things. But shortest distance can mean a lot of things! You can use breadth-first search to

La búsqueda en amplitud le permite encontrar la distancia más corta entre dos cosas. ¡Pero la distancia más corta puede significar muchas cosas! Puede utilizar la búsqueda en amplitud para

- Write a spellchecker (fewest edits from your misspelling to a real word—for example, READED → READER is one edit).

Escriba un corrector ortográfico (la menor cantidad de ediciones de su error ortográfico a una palabra real; por ejemplo, LEÍDO → LECTOR es una edición).

- Find the doctor closest to you in your network.

Encuentre el médico más cercano a usted en su red.

- Build a search engine crawler.

Cree un rastreador de motor de búsqueda.

Graph algorithms are some of the most useful algorithms I know. Make sure you read the next few chapters carefully—these are algorithms you'll be able to apply again and again.

Los algoritmos de gráficos son algunos de los algoritmos más útiles que conozco. Asegúrese de leer atentamente los

siguentes capítulos: estos son algoritmos que podrá aplicar una y otra vez.

## ***Introduction to graphs***

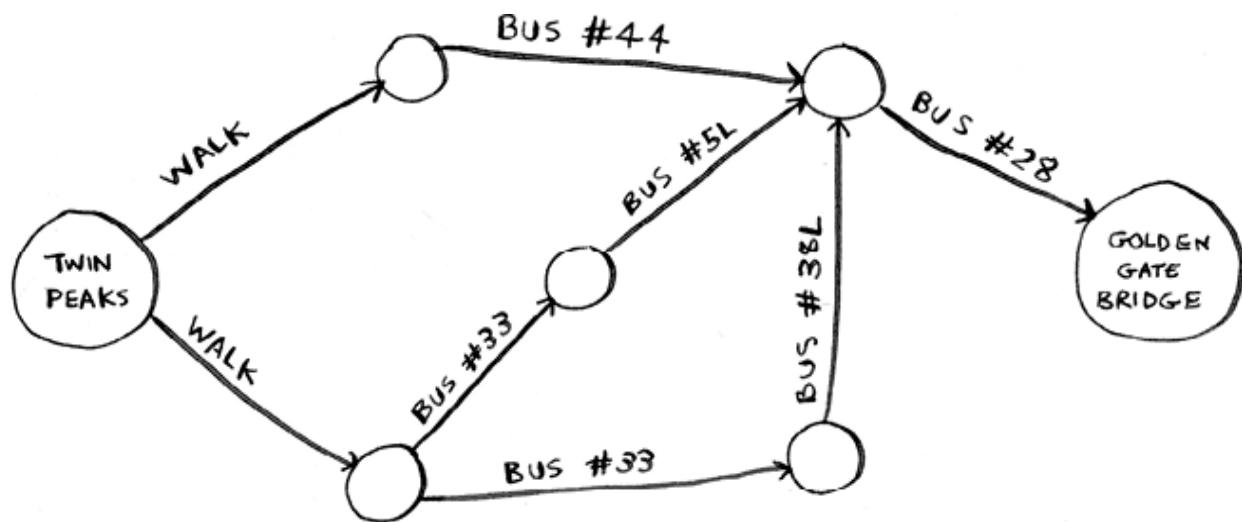
### ***Introducción a los gráficos.***



Suppose you're in San Francisco, and you want to go from Twin Peaks to the Golden Gate Bridge. You want to get there by bus, with the minimum number of transfers. Here are your options.

Supongamos que estás en San Francisco y quieres ir desde Twin Peaks hasta el puente Golden Gate. Quieres llegar en

autobús, con el mínimo de transbordos. Aquí están tus opciones.

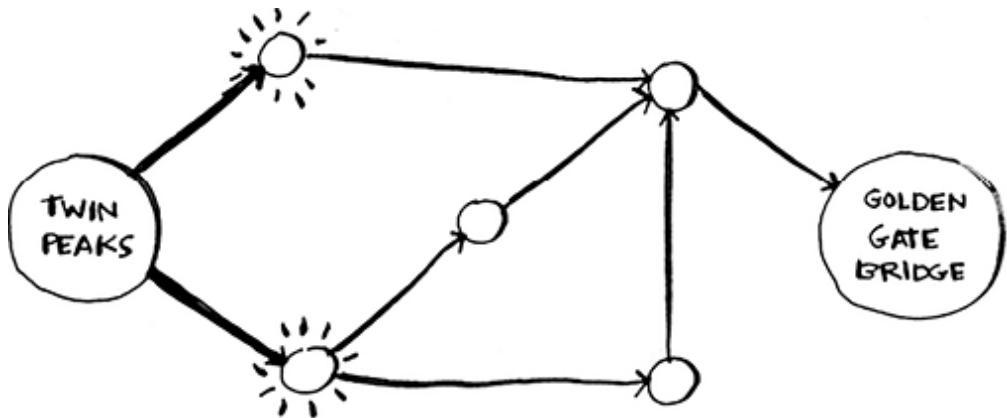


What's your algorithm to find the path with the fewest steps?

¿Cuál es tu algoritmo para encontrar el camino con la menor cantidad de pasos?

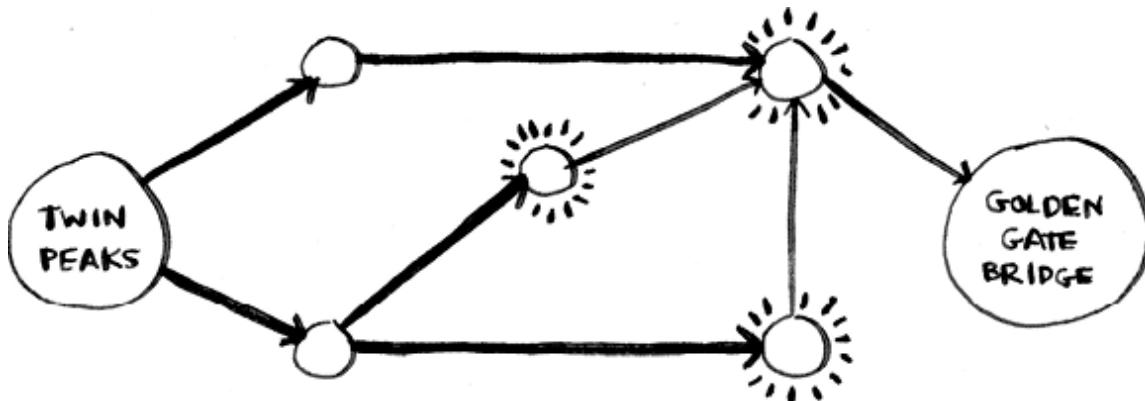
Well, can you get there in one step? Here are all the places you can get to in one step.

Bueno, ¿puedes llegar allí en un solo paso? Aquí tienes todos los lugares a los que puedes llegar en un solo paso.



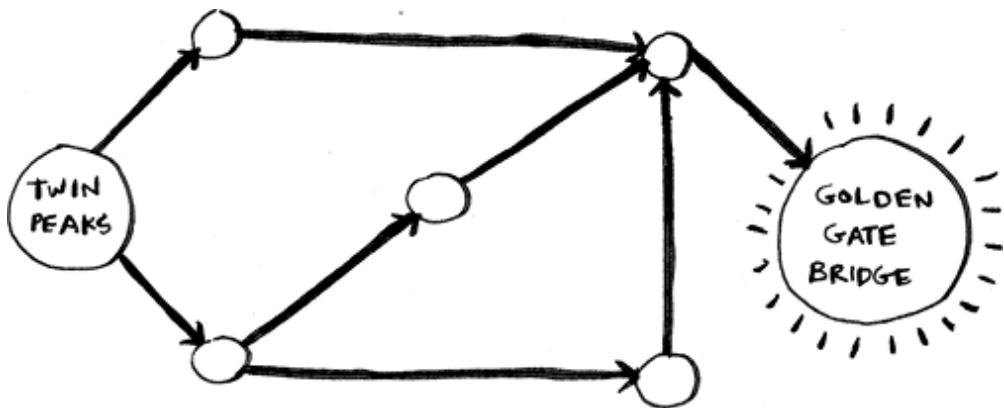
The bridge isn't highlighted; you can't get there in one step.  
Can you get there in two steps?

El puente no está resaltado; no puedes llegar allí en un solo paso. ¿Puedes llegar allí en dos pasos?



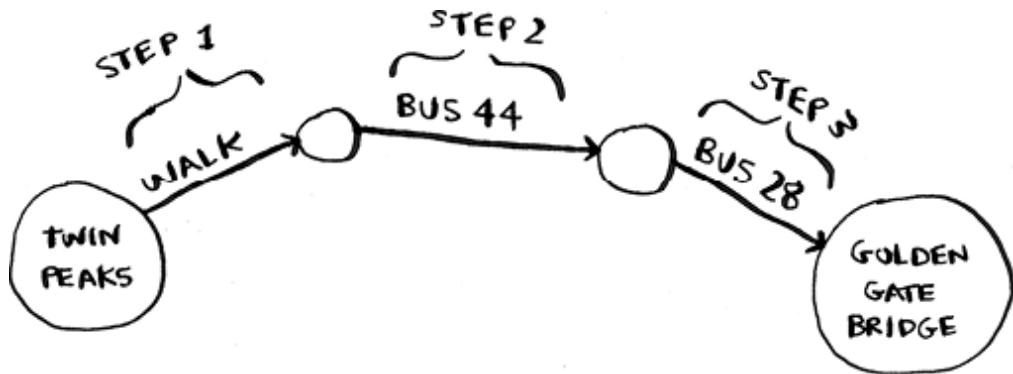
Again, the bridge isn't there, so you can't get to the bridge in two steps. What about three steps?

Nuevamente, el puente no está allí, por lo que no puedes llegar al puente en dos pasos. ¿Qué tal tres pasos?



Aha! Now the Golden Gate Bridge shows up. So it takes three steps to get from Twin Peaks to the bridge using this route.

¡Ajá! Ahora aparece el puente Golden Gate. Entonces, se necesitan tres pasos para llegar desde Twin Peaks al puente usando esta ruta.



There are other routes that will get you to the bridge, too, but they're longer (four steps). The algorithm found that the shortest route to the bridge is three steps long. This type of problem is called a *shortest-path problem*. You're always trying to find the shortest something. It could be the shortest route to your friend's house. Or maybe you're browsing the web. Without you knowing it, the network is

looking for the shortest path between your computer and a website's server. The algorithm to solve a shortest-path problem is called *breadth-first search*.

Hay otras rutas que también te llevarán al puente, pero son más largas (cuatro pasos). El algoritmo descubrió que la ruta más corta hacia el puente tiene tres pasos. Este tipo de problema se denomina problema del camino más corto. Siempre estás tratando de encontrar algo más corto. Podría ser la ruta más corta a la casa de tu amigo. O tal vez estés navegando por la web. Sin que usted lo sepa, la red busca el camino más corto entre su computadora y el servidor de un sitio web. El algoritmo para resolver un problema de camino más corto se llama búsqueda en amplitud.

There are two steps to figuring out how to get from Twin Peaks to the Golden Gate Bridge:

Hay dos pasos para descubrir cómo llegar desde Twin Peaks al puente Golden Gate:

1. Model the problem as a graph.

Modele el problema como una gráfica.

2. Solve the problem using breadth-first search.

Resuelva el problema utilizando la búsqueda en amplitud.

Next I'll cover what graphs are. Then I'll go into breadth-first search in more detail.

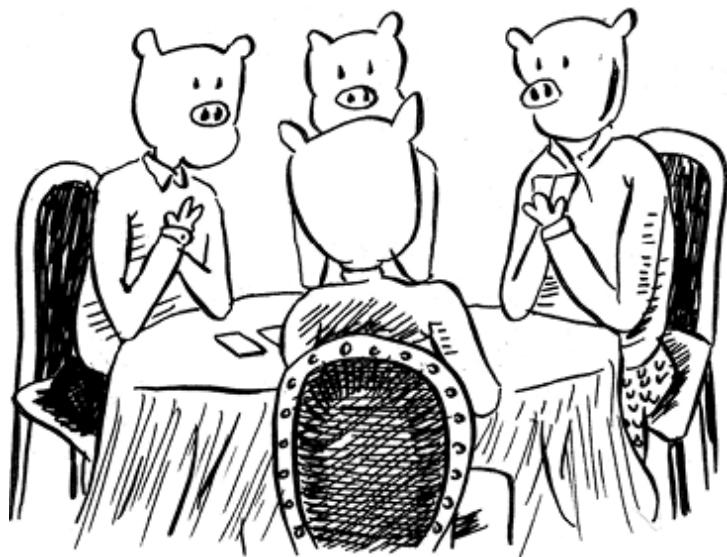
A continuación cubriré qué son los gráficos. Luego entraré en la búsqueda amplia con más detalle.

## **What is a graph?**

### **¿Qué es una gráfica?**

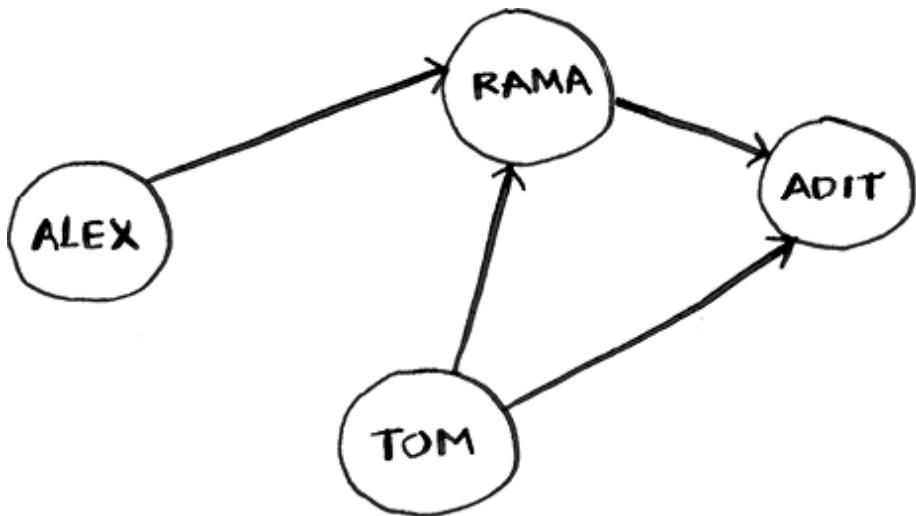
A graph models a set of connections. For example, suppose you and your friends are playing poker, and you want to model who owes whom money. Here's how you could say, "Alex owes Rama money."

Un gráfico modela un conjunto de conexiones. Por ejemplo, supongamos que usted y sus amigos están jugando al póquer y quiere modelar quién le debe dinero a quién. Así es como podrías decir: "Alex le debe dinero a Rama".



The full graph could look something like this.

El gráfico completo podría verse así.

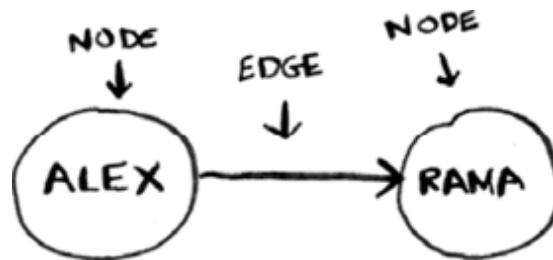


**Graph of people who owe other people poker money**

**Gráfico de personas que deben dinero al poker a otras personas**

Alex owes Rama money, Tom owes Adit money, and so on.  
Each graph is made up of *nodes* and *edges*.

Alex le debe dinero a Rama, Tom le debe dinero a Adit, etc.  
Cada gráfico se compone de nodos y aristas.



That's all there is to it! Graphs are made up of nodes and edges. A node can be directly connected to many other nodes. Those nodes are called *in-neighbors* or *out-neighbors*.

¡Eso es todo al respecto! Los gráficos se componen de nodos y aristas. Un nodo se puede conectar directamente a muchos otros nodos. Esos nodos se llaman vecinos internos o vecinos externos.

Since Alex is pointing to Rama, Alex is Rama's *in-neighbor*, and Rama is Alex's *out-neighbor*. This terminology can be confusing, so here's a diagram to help.

Dado que Alex señala a Rama, Alex es el vecino interno de Rama y Rama es el vecino externo de Alex. Esta terminología puede resultar confusa, por lo que aquí tienes un diagrama que te ayudará.



In the graph, Adit isn't Alex's in-neighbor or out-neighbor because they aren't directly connected. But Adit is Rama's and Tom's out-neighbor.

En el gráfico, Adit no es vecino interno ni externo de Alex porque no están conectados directamente. Pero Adit es el vecino externo de Rama y Tom.

Graphs are a way to model how different things are connected to one another. Now let's see breadth-first search in action.

Los gráficos son una forma de modelar cómo se conectan diferentes cosas entre sí. Ahora veamos la búsqueda en amplitud en acción.

## **Breadth-first search**

### **Búsqueda en amplitud**

We looked at a search algorithm in chapter 1: binary search. Breadth-first search is a different kind of search algorithm: one that runs on graphs. It can help answer two types of questions:

Analizamos un algoritmo de búsqueda en el capítulo 1: búsqueda binaria. La búsqueda en amplitud es un tipo diferente de algoritmo de búsqueda: uno que se ejecuta en gráficos. Puede ayudar a responder dos tipos de preguntas:

- Question type 1: Is there a path from node A to node B?  
Tipo de pregunta 1: ¿Existe una ruta desde el nodo A al nodo B?
- Question type 2: What is the shortest path from node A to node B?  
Tipo de pregunta 2: ¿Cuál es el camino más corto desde el nodo A al nodo B?

You already saw breadth-first search once when you calculated the shortest route from Twin Peaks to the Golden Gate Bridge. That was a question of type 2: "What is the

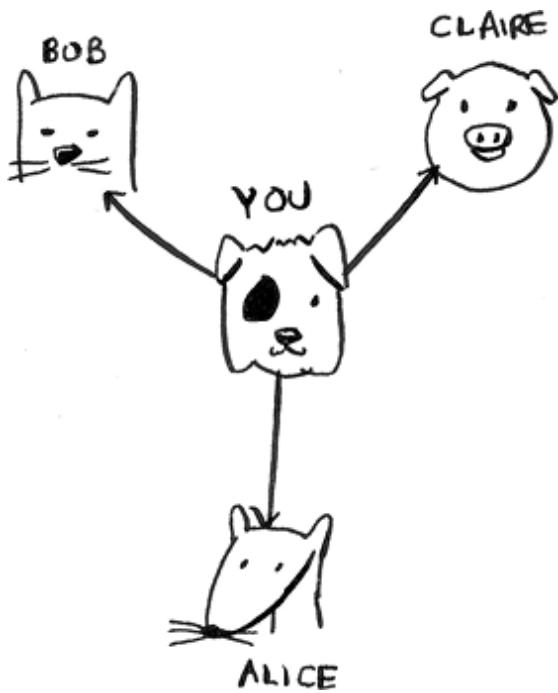
shortest path?" Now let's look at the algorithm in more detail. You'll ask a question of type 1: "Is there a path?"

Ya vio una vez la búsqueda en amplitud cuando calculó la ruta más corta desde Twin Peaks hasta el puente Golden Gate. Esa era una pregunta del tipo 2: "¿Cuál es el camino más corto?" Ahora veamos el algoritmo con más detalle. Harás una pregunta del tipo 1: "¿Existe un camino?"



Suppose you're the proud owner of a mango farm. You're looking for a mango seller who can sell your mangoes. Are you connected to a mango seller on Facebook? Well, you can search through your friends.

Supongamos que eres el orgulloso propietario de una granja de mangos. Estás buscando un vendedor de mangos que pueda vender sus mangos. ¿Estás conectado con un vendedor de mangos en Facebook? Bueno, puedes buscar entre tus amigos.



This search is pretty straightforward.

Esta búsqueda es bastante sencilla.

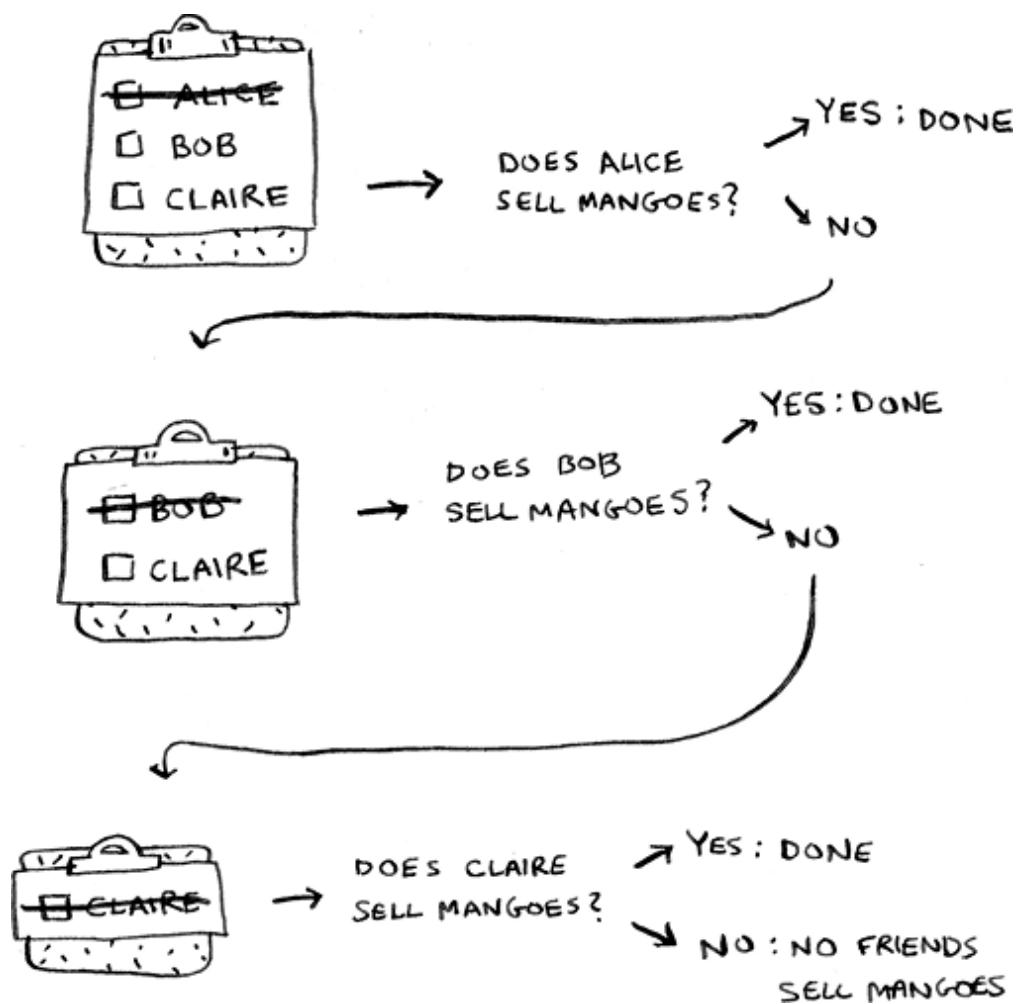
First, make a list of friends to search.

Primero, haz una lista de amigos para buscar.



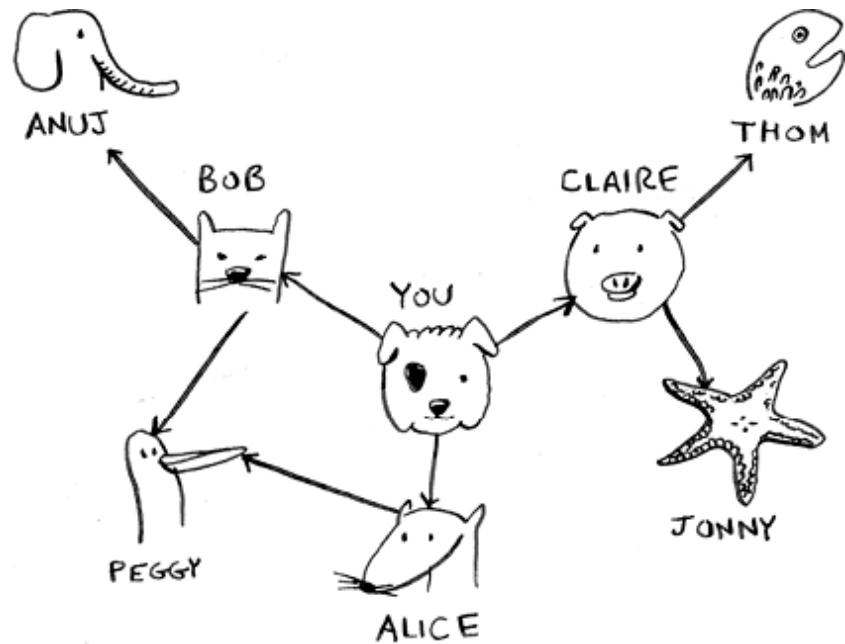
Now, go to each person in the list and check whether that person sells mangoes.

Ahora, ve a cada persona de la lista y comprueba si esa persona vende mangos.



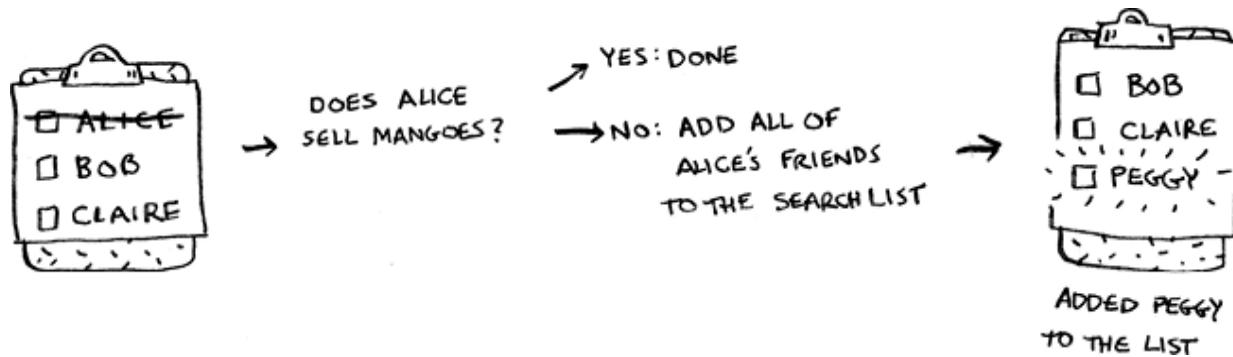
Suppose none of your friends are mango sellers. Now you have to search through your friends' friends.

Supongamos que ninguno de tus amigos es vendedor de mangos. Ahora tienes que buscar entre los amigos de tus amigos.



Each time you search for someone from the list, add all of their friends to the list.

Cada vez que busques a alguien de la lista, agrega todos sus amigos a la lista.



This way, you not only search your friends, but you search their friends, too. Remember, the goal is to find one mango seller in your network. So if Alice isn't a mango seller, you add her friends to the list, too. That means you'll eventually search her friends—and then their friends, and so on. With

this algorithm, you'll search your entire network until you come across a mango seller. This algorithm is breadth-first search.

De esta manera, no sólo buscas a tus amigos, sino que también buscas a sus amigos. Recuerde, el objetivo es encontrar un vendedor de mango en su red. Entonces, si Alice no es vendedora de mangos, también agregas a sus amigos a la lista. Eso significa que eventualmente buscarás a sus amigos, y luego a sus amigos, y así sucesivamente. Con este algoritmo, buscarás en toda tu red hasta encontrar un vendedor de mango. Este algoritmo es una búsqueda de amplitud.

## Finding the shortest path

### Encontrar el camino más corto

As a recap, these are the two questions that breadth-first search can answer for you:

A modo de resumen, estas son las dos preguntas que la búsqueda en amplitud puede responder:

- Question type 1: Is there a path from node A to node B? (Is there a mango seller in your network?)

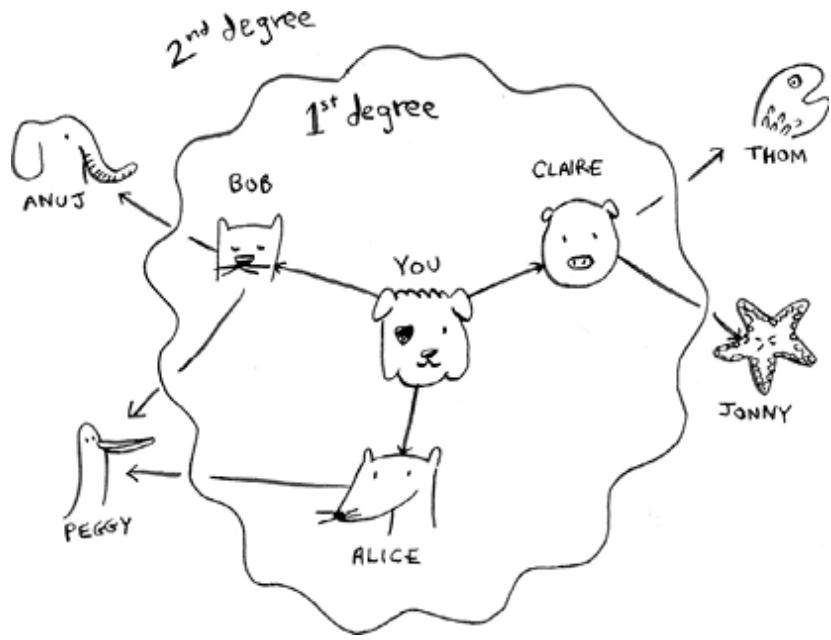
Tipo de pregunta 1: ¿Existe una ruta desde el nodo A al nodo B? (¿Hay algún vendedor de mango en su red?)

- Question type 2: What is the shortest path from node A to node B? (Who is the closest mango seller?)

Tipo de pregunta 2: ¿Cuál es el camino más corto desde el nodo A al nodo B? (¿Quién es el vendedor de mangos más cercano?)

You saw how to answer question 1; now let's try to answer question 2. Can you find the closest mango seller? For example, your friends are first-degree connections, and their friends are second-degree connections.

Viste cómo responder la pregunta 1; Ahora intentemos responder la pregunta 2. ¿Puedes encontrar el vendedor de mangos más cercano? Por ejemplo, tus amigos son conexiones de primer grado y sus amigos son conexiones de segundo grado.



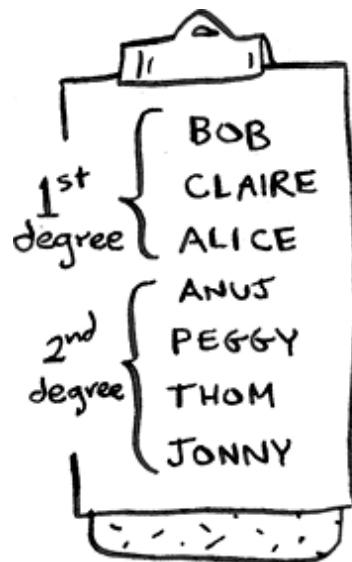
You'd prefer a first-degree connection to a second-degree connection, a second-degree connection to a third-degree connection, and so on. So you shouldn't search any second-degree connections before you make sure you don't have a first-degree connection who is a mango seller. Well, breadth-first search already does this! The way breadth-first search works is that the search radiates out from the starting point. So you'll check first-degree connections before second-degree connections. Pop quiz: Who will be checked first, Claire or Anuj? Answer: Claire is a first-degree connection, and Anuj is a second-degree connection, so Claire will be checked before Anuj.

Preferiría una conexión de primer grado a una conexión de segundo grado, una conexión de segundo grado a una conexión de tercer grado, y así sucesivamente. Por lo tanto, no debe buscar ninguna conexión de segundo grado antes de asegurarse de no tener una conexión de primer grado que sea vendedor de mango. Bueno, la búsqueda en

amplitud ya hace esto! La forma en que funciona la búsqueda en amplitud es que la búsqueda se irradia desde el punto de partida. Por lo tanto, comprobará las conexiones de primer grado antes que las de segundo grado. Examen sorpresa: ¿Quién será evaluado primero, Claire o Anuj? Respuesta: Claire es una conexión de primer grado y Anuj es una conexión de segundo grado, por lo que Claire será revisada antes que Anuj.

Another way to see this is that first-degree connections are added to the search list before second-degree connections.

Otra forma de ver esto es que las conexiones de primer grado se agregan a la lista de búsqueda antes que las conexiones de segundo grado.



You just go down the list and check people to see whether each one is a mango seller. The first-degree connections will be searched before the second-degree connections, so you'll

find the mango seller closest to you. Breadth-first search not only finds a path from A to B; it also finds the shortest path.

Simplemente recorra la lista y verifique a las personas para ver si cada una es vendedora de mango. Las conexiones de primer grado se buscarán antes que las de segundo grado, por lo que encontrará al vendedor de mango más cercano a usted. La búsqueda en amplitud no sólo encuentra un camino de A a B; también encuentra el camino más corto.

Notice that this only works if you search people in the same order in which they're added. That is, if Claire was added to the list before Anuj, Claire needs to be searched before Anuj. What happens if you search Anuj before Claire, and they're both mango sellers? Well, Anuj is a second-degree contact, and Claire is a first-degree contact. You end up with a mango seller who isn't the closest to you in your network. So you need to search people in the order that they're added. There's a data structure for this: it's called a *queue*.

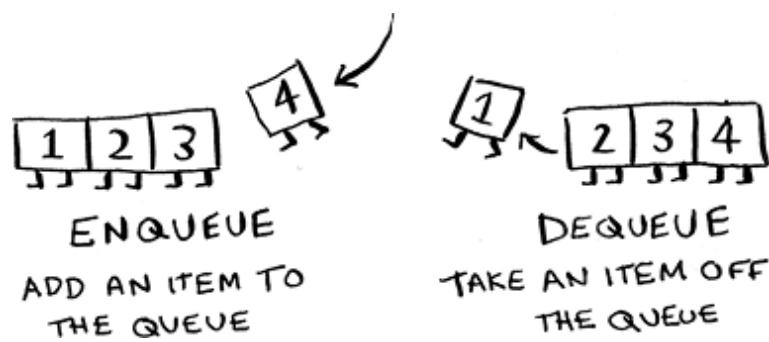
Tenga en cuenta que esto sólo funciona si busca personas en el mismo orden en que se agregaron. Es decir, si Claire fue agregada a la lista antes que Anuj, Claire debe ser buscada antes que Anuj. ¿Qué pasa si buscas a Anuj antes que a Claire y ambos son vendedores de mangos? Bueno, Anuj es un contacto de segundo grado y Claire es un contacto de primer grado. Terminas con un vendedor de mangos que no es el más cercano a ti en tu red. Por lo tanto, debes buscar personas en el orden en que se agregan. Existe una estructura de datos para esto: se llama cola.

## Queues

## Colas

A queue works exactly like it does in real life. Suppose you and your friend are queueing up at the bus stop. If you're before them in the queue, you get on the bus first. A queue works the same way. Queues are similar to stacks. You can't access random elements in the queue. Instead, there are only two operations, *enqueue* and *dequeue*.

Una cola funciona exactamente igual que en la vida real. Suponga que usted y su amigo están haciendo cola en la parada de autobús. Si estás delante de ellos en la cola, te subes primero al autobús. Una cola funciona de la misma manera. Las colas son similares a las pilas. No puedes acceder a elementos aleatorios en la cola. En cambio, solo hay dos operaciones: poner en cola y quitar de la cola.

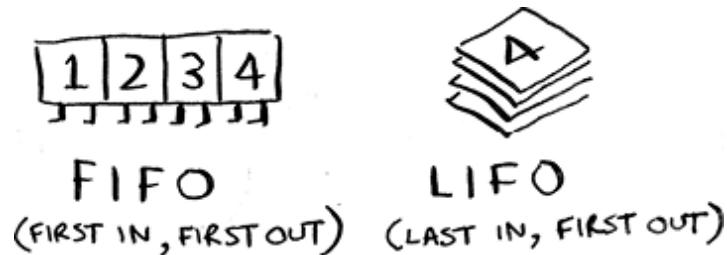


If you enqueue two items to the list, the first item you added will be dequeued before the second item. You can use this for your search list! People who are added to the list first will be dequeued and searched first.

Si pone en cola dos elementos de la lista, el primer elemento que agregó será retirado de la cola antes que el segundo. ¡Puedes usar esto para tu lista de búsqueda! Las personas que se agreguen primero a la lista serán retiradas de la cola y buscadas primero.

The queue is called a *FIFO* data structure: first in, first out. In contrast, a stack is a *LIFO* data structure: last in, first out.

La cola se denomina estructura de datos FIFO: primero en entrar, primero en salir. Por el contrario, una pila es una estructura de datos LIFO: el último en entrar, el primero en salir.



Now that you know how a queue works, let's implement breadth-first search!

Ahora que sabes cómo funciona una cola, implementemos la búsqueda en amplitud!

## EXERCISES

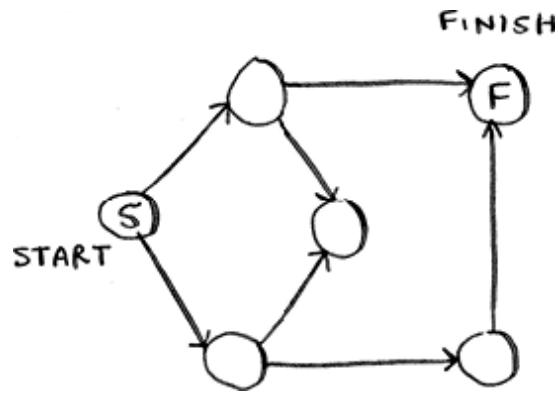
## EJERCICIOS

Run the breadth-first search algorithm on each of these graphs to find the solution.

Ejecute el algoritmo de búsqueda en amplitud en cada uno de estos gráficos para encontrar la solución.

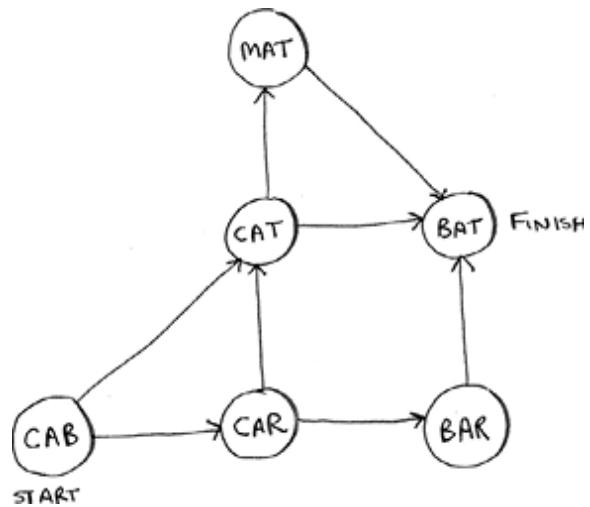
**6.1** Find the length of the shortest path from start to finish.

6.1 Encuentre la longitud del camino más corto de principio a fin.



**6.2** Find the length of the shortest path from "cab" to "bat."

6.2 Encuentre la longitud del camino más corto desde "taxi" hasta "bat".

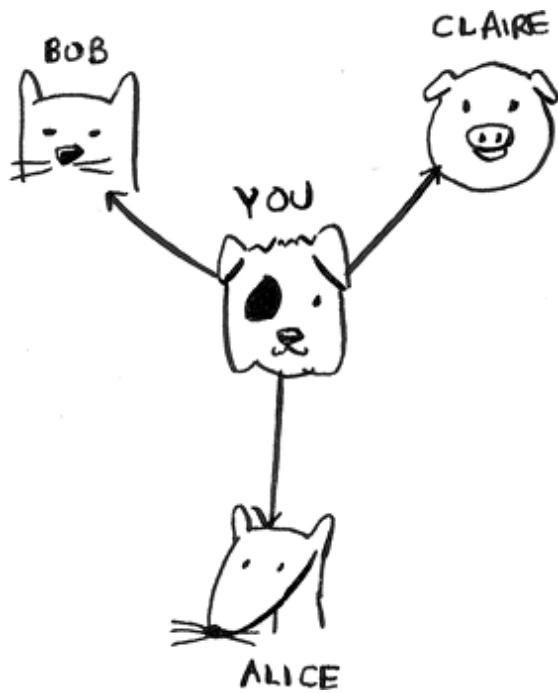


## ***Implementing the graph***

## ***Implementando el gráfico***

First, you need to implement the graph in code. A graph consists of several nodes.

Primero, necesitas implementar el gráfico en el código. Un gráfico consta de varios nodos.

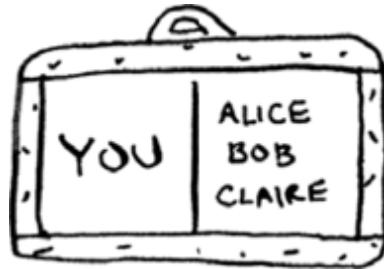


And each node is connected to other nodes. How do you express a relationship like “you → bob”? Luckily, you know a data structure that lets you express relationships: *a hash table!*

Y cada nodo está conectado a otros nodos. ¿Cómo se expresa una relación como “tú → bob”? Afortunadamente, conoces una estructura de datos que te permite expresar relaciones: iuna tabla hash!

Remember, a hash table allows you to map a key to a value. In this case, you want to map a node to all of its out-neighbors.

Recuerde, una tabla hash le permite asignar una clave a un valor. En este caso, desea asignar un nodo a todos sus vecinos.



Here's how you'd write it in Python:

Así es como lo escribirías en Python:

```
graph = {}  
graph["you"] = ["alice", "bob", "claire"]
```

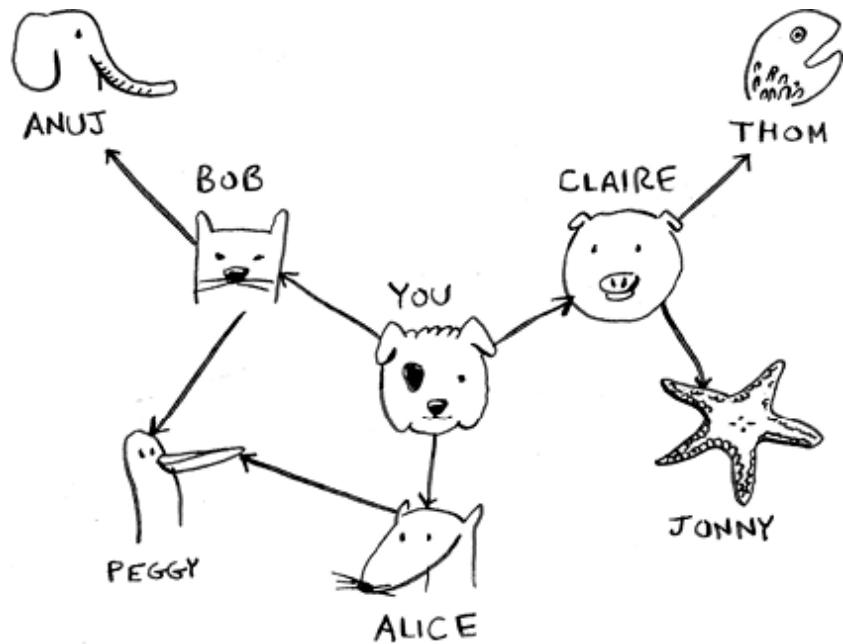
Notice that "you" is mapped to an array. So `graph["you"]` will give you an array of all the out-neighbors of "you."

Remember that the out-neighbors are the nodes that the "you" node points to.

Observe que "usted" está asignado a una matriz. Entonces `graph["you"]` le dará una serie de todos los vecinos externos de "usted". Recuerde que los vecinos externos son los nodos a los que apunta el nodo "usted".

A graph is just a bunch of nodes and edges, so this is all you need to have a graph in Python. What about a bigger graph like this one?

Un gráfico es solo un conjunto de nodos y aristas, por lo que esto es todo lo que necesitas para tener un gráfico en Python. ¿Qué tal un gráfico más grande como este?



Here it is as Python code:

Aquí está como código Python:

```

graph = {}
graph["you"] = ["alice", "bob", "claire"]
graph["bob"] = ["anuj", "peggy"]
graph["alice"] = ["peggy"]
graph["claire"] = ["thom", "jonny"]
graph["anuj"] = []
graph["peggy"] = []
graph["thom"] = []
graph["jonny"] = []

```

Pop quiz: Does it matter what order you add the key/value pairs in? Does it matter if you write

Prueba sorpresa: ¿Importa en qué orden agregue los pares clave/valor? ¿Importa si escribes?

```

graph["claire"] = ["thom", "jonny"]
graph["anuj"] = []

```

instead of

en lugar de

```
graph["anuj"] = []
graph["claire"] = ["thom", "jonny"]
```

Think back to the previous chapter. Answer: It doesn't matter. Hash tables have no ordering, so it doesn't matter what order you add key/value pairs in.

Piensa en el capítulo anterior. Respuesta: No importa. Las tablas hash no tienen orden, por lo que no importa en qué orden agregue los pares clave/valor.

Anuj, Peggy, Thom, and Jonny don't have any out-neighbors. They have in-neighbors since they have arrows pointing to them, but no arrows point from them to someone else. This is called a *directed graph*: the relationship is only one way. An undirected graph doesn't have any arrows. For example, both of these graphs are equal.

Anuj, Peggy, Thom y Jonny no tienen vecinos. Tienen vecinos internos ya que tienen flechas que apuntan hacia ellos, pero ninguna flecha apunta desde ellos hacia otra persona. A esto se le llama gráfico dirigido: la relación es sólo en un sentido. Un gráfico no dirigido no tiene flechas. Por ejemplo, ambas gráficas son iguales.



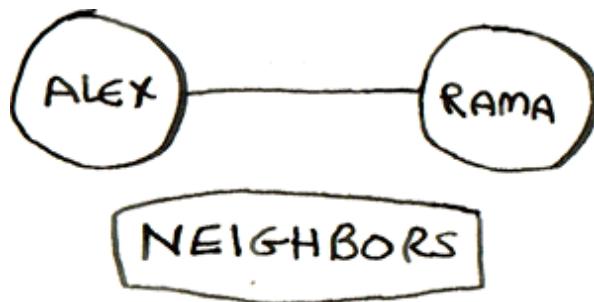
DIRECTED  
GRAPH



UNDIRECTED  
GRAPH

If you have an undirected graph, you can forget the terms in-neighbor and out-neighbor and use the simpler term *neighbor*.

Si tiene un gráfico no dirigido, puede olvidar los términos vecino interno y vecino externo y usar el término vecino más simple.

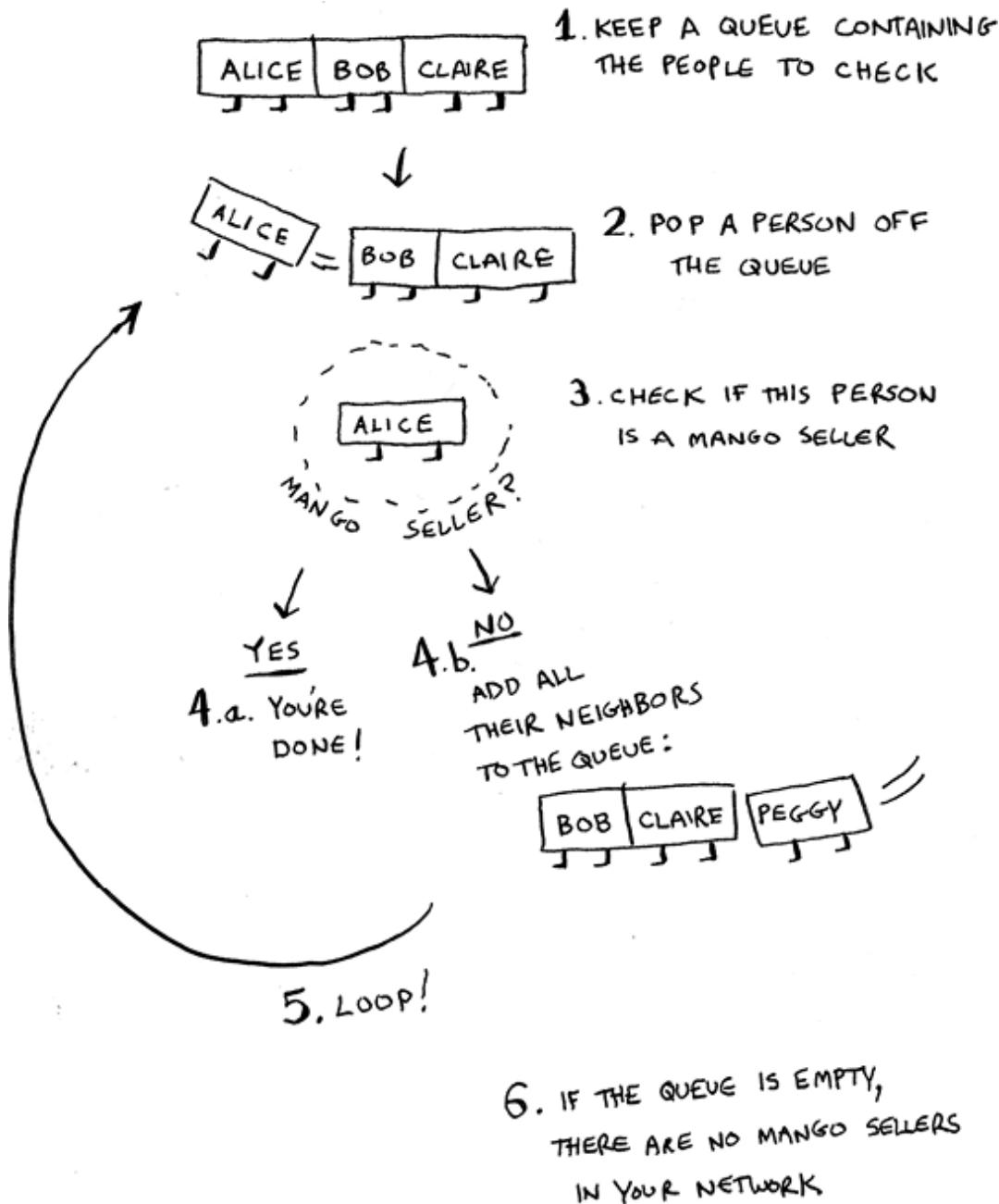


## ***Implementing the algorithm***

## ***Implementando el algoritmo***

To recap, here's how the implementation will work.

En resumen, así es como funcionará la implementación.



**NOTE** When updating queues, I use the terms `enqueue` and `dequeue`. But sometimes you'll see people using different terms. Python uses `append` and `popleft` instead of `enqueue` and `dequeue`.

Nota Al actualizar colas, utilizo los términos poner en cola y sacar de cola. Pero a veces verás personas que usan términos diferentes. Python usa `append` y `popleft` en lugar de poner y quitar de la cola.

Make a queue to start. In Python, you use the double-ended queue (`deque`) function for this:

Haz cola para empezar. En Python, usas la función de cola de doble extremo (`deque`) para esto:

```
from collections import deque  
search_queue = deque() ①  
search_queue += graph["you"] ②
```

① Creates a new queue

① Crea una nueva cola

② Adds all of your out-neighbors to the search queue

② Agrega todos sus vecinos externos a la cola de búsqueda

Remember, `graph["you"]` will give you a list of all your out-neighbors, like `["alice", "bob", "claire"]`. Those all get added to the search queue.

Recuerde, `graph["you"]` le dará una lista de todos sus vecinos externos, como `["alice", "bob", "claire"]`. Todos ellos se agregan a la cola de búsqueda.



Let's see the rest:

Veamos el resto:

```

while search_queue:
    person = search_queue.popleft()
    if person_is_seller(person):
        print(person + " is a mango seller!")
        return True
    else:
        search_queue += graph[person]
return False

```

- ① While the queue isn't empty . . .
- ① Mientras la cola no esté vacía . . .
- ② . . . grabs the first person off the queue.
- ② . . . Agarra a la primera persona de la cola.
- ③ Checks whether the person is a mango seller
- ③ Comprueba si la persona es vendedora de mangos.
- ④ Yes, they're a mango seller.
- ④ Sí, es vendedor de mangos.
- ⑤ No, they aren't. Add all of this person's friends to the search queue.
- ⑤ No, no lo son. Agrega todos los amigos de esta persona a la cola de búsqueda.
- ⑥ If you reached here, no one in the queue is a mango seller.
- ⑥ Si llegaste hasta aquí, nadie en la cola es vendedor de mangos.

One final thing: you still need a `person_is_seller` function to tell you when someone is a mango seller. Here's one:

Una última cosa: todavía necesitas una función `person_is_seller` para saber cuándo alguien es vendedor de mangos. Aquí hay uno:

```

def person_is_seller(name):
    return name[-1] == 'm'

```

This function checks whether the person's name ends with the letter *m*. If it does, they're a mango seller. Kind of a silly

way to do it, but it'll do for this example. Now let's see the breadth-first search in action.

Esta función comprueba si el nombre de la persona termina con la letra m. Si es así, es un vendedor de mangos. Es una manera un poco tonta de hacerlo, pero servirá para este ejemplo. Ahora veamos la búsqueda en amplitud en acción.



...etc...

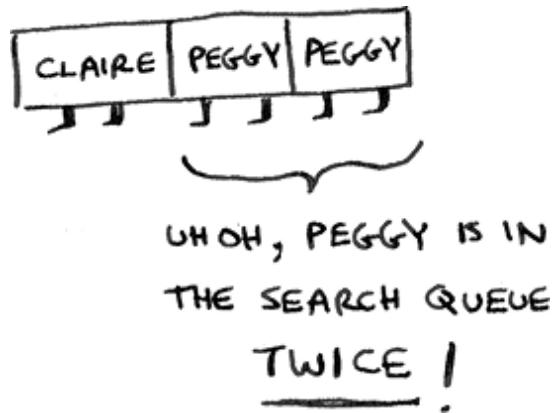
And so on. The algorithm will keep going until either a mango seller is found or the queue becomes empty, in which case there is no mango seller.

Etcétera. El algoritmo continuará hasta que se encuentre un vendedor de mangos o la cola quede vacía, en cuyo caso no

habrá ningún vendedor de mangos.

Alice and Bob share a friend, Peggy. So Peggy will be added to the queue twice: once when you add Alice's friends and again when you add Bob's friends. You'll end up with two Peggys in the search queue.

Alice y Bob comparten una amiga, Peggy. Entonces, Peggy se agregará a la cola dos veces: una vez cuando agregues a los amigos de Alice y otra vez cuando agregues a los amigos de Bob. Terminarás con dos Peggys en la cola de búsqueda.



But you only need to check Peggy once to see whether she's a mango seller. If you check her twice, you're doing unnecessary, extra work. So once you search a person, you should mark that person as searched and not search them again.

Pero sólo necesitas comprobar a Peggy una vez para ver si es vendedora de mangos. Si la revisas dos veces, estás haciendo un trabajo extra innecesario. Entonces, una vez que busques a una persona, debes marcarla como buscada y no volver a buscarla.

If you don't do this, you could also end up in an infinite loop. Suppose the mango seller graph looked like this.

Si no haces esto, también podrías terminar en un bucle infinito. Supongamos que el gráfico del vendedor de mangos se viera así.



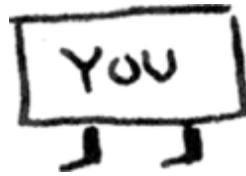
To start, the search queue contains all of your out-neighbors.

Para empezar, la cola de búsqueda contiene todos sus vecinos externos.



Now you check Peggy. She isn't a mango seller, so you add all her out-neighbors to the search queue.

Ahora revisa a Peggy. Ella no es vendedora de mangos, por lo que agregas a todos sus vecinos externos a la cola de búsqueda.



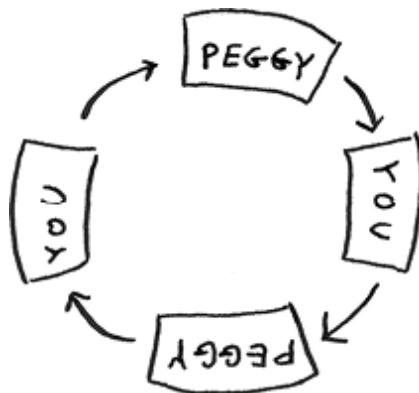
Next, you check yourself. You're not a mango seller, so you add all your out-neighbors to the search queue.

A continuación, usted mismo se controla. No eres un vendedor de mangos, por lo que agregas a todos tus vecinos externos a la cola de búsqueda.



And so on. This will be an infinite loop because the search queue will keep going from you to Peggy.

Etcétera. Esto será un bucle infinito porque la cola de búsqueda seguirá yendo de ti a Peggy.



Before checking a person, it's important to make sure they haven't been checked already. To do that, you'll keep a set of people you've already checked.

Antes de controlar a una persona, es importante asegurarse de que no la hayan controlado ya. Para hacer eso, mantendrás un conjunto de personas que ya has verificado.



Here's the final code for breadth-first search, taking that into account:

Aquí está el código final para la búsqueda en amplitud, teniendo eso en cuenta:

```
def search(name):
    search_queue = deque()
    search_queue += graph[name]
```

```

searched = set()                                ①
while search_queue:
    person = search_queue.popleft()
    if not person in searched:                  ②
        if person_is_seller(person):
            print(person + " is a mango seller!")
            return True
        else:
            search_queue += graph[person]
            searched.add(person)                 ③
    return False

search("you")

```

- ① This set is how you keep track of which people you've searched before.
- ① Este conjunto es la forma en que realiza un seguimiento de las personas que ha buscado antes.
- ② Only search this person if you haven't already searched them.
- ② Busque a esta persona solo si aún no la ha buscado.
- ③ Marks this person as searched
- ③ Marca a esta persona como buscada

Try running this code yourself. Maybe try changing the `person_is_seller` function to something more meaningful and see if it prints what you expect.

Intente ejecutar este código usted mismo. Tal vez intente cambiar la función `person_is_seller` a algo más significativo y vea si imprime lo que espera.

## Running time

### Tiempo de ejecución

If you search your entire network for a mango seller, that means you'll follow each edge (remember, an edge is the

arrow or connection from one person to another). So the running time is at least  $O(\text{number of edges})$ .

Si busca en toda su red un vendedor de mango, eso significa que seguirá cada borde (recuerde, un borde es la flecha o la conexión de una persona a otra). Entonces el tiempo de ejecución es al menos  $O(\text{número de aristas})$ .

You also keep a queue of every person to search. Adding one person to the queue takes constant time:  $O(1)$ . Doing this for every person will take  $O(\text{number of people})$  total.

Breadth-first search takes  $O(\text{number of people} + \text{number of edges})$ , and it's more commonly written as  $O(V+E)$  ( $V$  for number of vertices;  $E$  for number of edges).

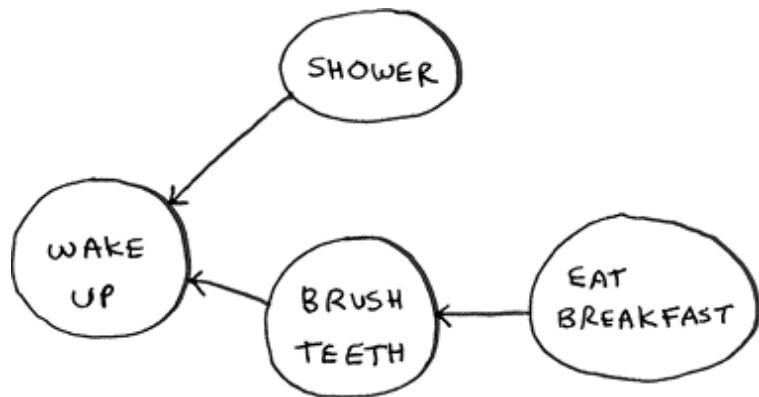
También mantienes una cola de cada persona para buscar. Agregar una persona a la cola lleva un tiempo constante:  $O(1)$ . Hacer esto para cada persona obtendrá  $O(\text{número de personas})$  en total. La búsqueda en amplitud toma  $O(\text{número de personas} + \text{número de aristas})$  y se escribe más comúnmente como  $O(V+E)$  ( $V$  para número de vértices;  $E$  para número de aristas).

## **EXERCISES**

## **EJERCICIOS**

Here's a small graph of my morning routine.

Aquí tenéis un pequeño gráfico de mi rutina matutina.



It tells you that I can't eat breakfast until I've brushed my teeth. So "eat breakfast" *depends on* "brush teeth."

Te dice que no puedo desayunar hasta que me haya lavado los dientes. Entonces "desayunar" depende de "cepillarse los dientes".

On the other hand, showering doesn't depend on brushing my teeth because I can shower before I brush my teeth.  
From this graph, you can make a list of the order in which I need to do my morning routine:

Por otro lado, ducharme no depende de cepillarme los dientes porque puedo ducharme antes de cepillarme los dientes. A partir de este gráfico, puedes hacer una lista del orden en el que necesito hacer mi rutina matutina:

1. Wake up.  
Despertar.
2. Shower.  
Ducha.
3. Brush teeth.  
Cepillar los dientes.

4. Eat breakfast.

Desayunar.

Note that “shower” can be moved around, so this list is also valid:

Tenga en cuenta que la “ducha” se puede mover, por lo que esta lista también es válida:

1. Wake up.

Despertar.

2. Brush teeth.

Cepillar los dientes.

3. Shower.

Ducha.

4. Eat breakfast.

Desayunar.

**6.3** For these three lists, mark whether each one is valid or invalid.

6.3 Para estas tres listas, marque si cada una es válida o no válida.

A.

1. WAKE UP
2. SHOWER
3. EAT BREAKFAST
4. BRUSH TEETH

B.

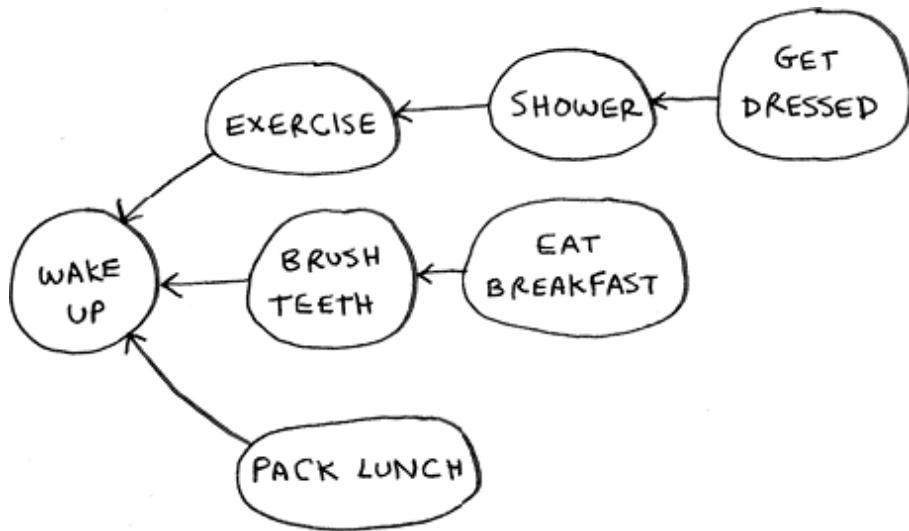
1. WAKE UP
2. BRUSH TEETH
3. EAT BREAKFAST
4. SHOWER

C.

1. SHOWER
2. WAKE UP
3. BRUSH TEETH
4. EAT BREAKFAST

**6.4** Here's a larger graph. Make a valid list for this graph.

6.4 Aquí hay un gráfico más grande. Haz una lista válida para este gráfico.



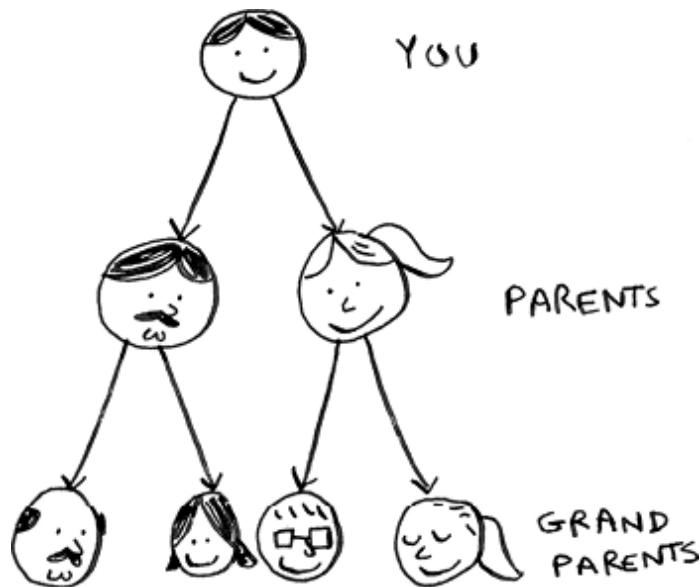
You could say that this list is sorted, in a way. If task A depends on task B, task A shows up later in the list. This is called a *topological sort*, and it's a way to make an ordered list out of a graph. Suppose you're planning a wedding and have a large graph full of tasks to do, and you're not sure where to start. You

could *topologically sort* the graph and get a list of tasks to do in order.

Se podría decir que esta lista está ordenada, en cierto modo. Si la tarea A depende de la tarea B, la tarea A aparece más adelante en la lista. Esto se llama clasificación topológica y es una forma de hacer una lista ordenada a partir de un gráfico. Supongamos que estás planeando una boda y tienes un gráfico grande lleno de tareas por hacer y no estás seguro de por dónde empezar. Podría ordenar topológicamente el gráfico y obtener una lista de tareas por realizar en orden.

Suppose you have a family tree.

Supongamos que tiene un árbol genealógico.

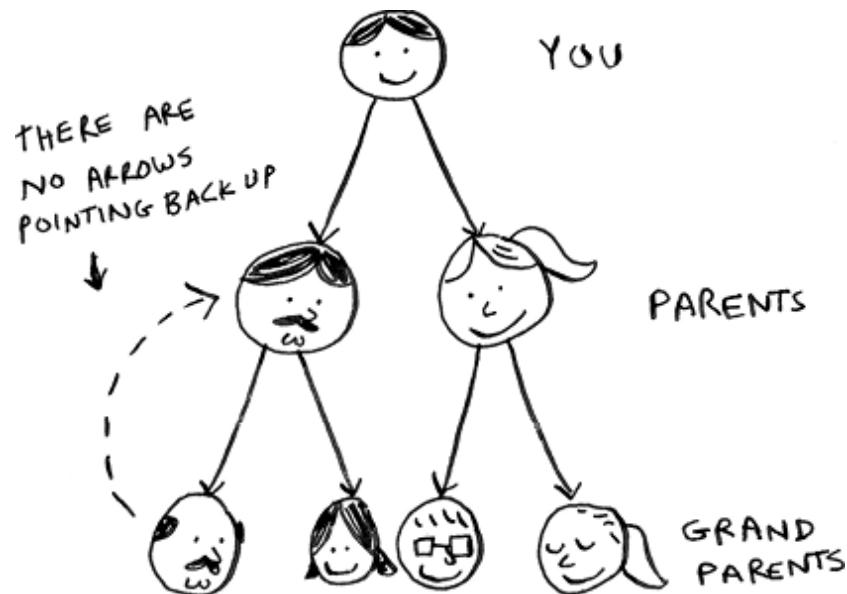


This is a graph, because you have nodes (the people) and edges.

Este es un gráfico, porque tienes nodos (las personas) y aristas.

The edges point to the nodes' parents. But all the edges go down—it wouldn't make sense for a family tree to have an edge pointing back up! That would be meaningless—your dad can't be your grandfather's dad!

Los bordes apuntan a los padres de los nodos. Pero todos los bordes van hacia abajo: no tendría sentido que un árbol genealógico tuviera un borde apuntando hacia arriba! Eso no tendría sentido: tu papá no puede ser el papá de tu abuelo!



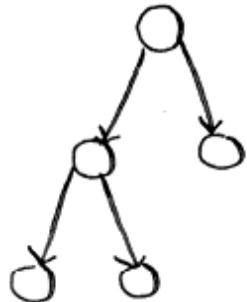
This is called a *tree*. A tree is a special type of graph where no edges ever point back.

A esto se le llama árbol. Un árbol es un tipo especial de gráfico en el que ningún borde apunta hacia atrás.

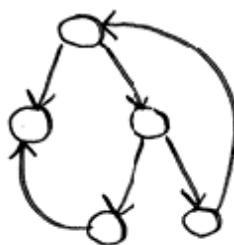
**6.5** Which of the following graphs are also trees?

6.5 ¿Cuáles de los siguientes gráficos también son árboles?

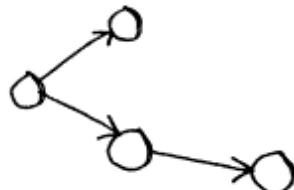
A.



B.



C.



## **Recap**

## **Resumen**

- Breadth-first search tells you if there's a path from A to B.

La búsqueda en amplitud le indica si hay una ruta de A a B.

- If there's a path, breadth-first search will find the shortest path.

Si hay un camino, la búsqueda en amplitud encontrará el camino más corto.

- If you have a problem like "find the shortest X," try modeling your problem as a graph and use breadth-first search to solve it.

Si tiene un problema como "encontrar la X más corta", intente modelar su problema como un gráfico y utilice la búsqueda en amplitud para resolverlo.

- A directed graph has arrows, and the relationship follows the direction of the arrow (rama → adit means "rama owes adit money").

Un gráfico dirigido tiene flechas y la relación sigue la dirección de la flecha (rama → adit significa "rama le debe dinero a adit").

- Undirected graphs don't have arrows, and the relationship goes both ways (ross— rachel means "ross dated rachel and rachel dated ross").

Los gráficos no dirigidos no tienen flechas y la relación es en ambos sentidos (ross—rachel significa "ross salió con rachel y rachel salió con ross").

- Queues are FIFO (first in, first out).

Las colas son FIFO (primero en entrar, primero en salir).

- Stacks are LIFO (last in, first out).

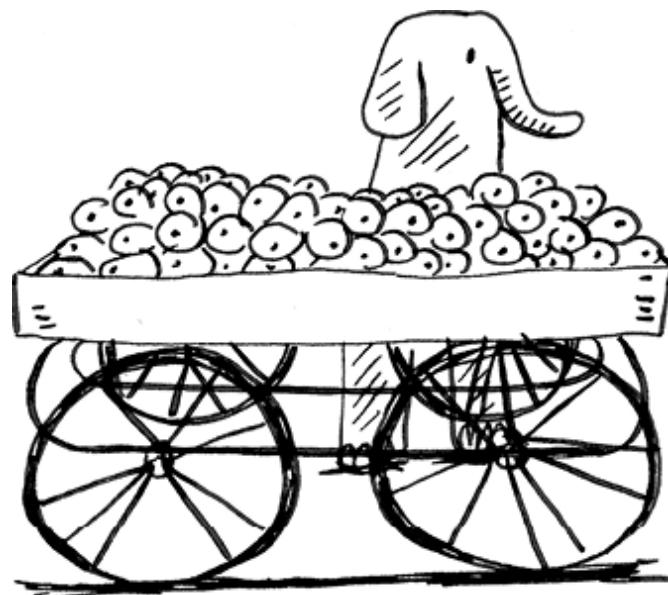
Las pilas son LIFO (último en entrar, primero en salir).

- You need to check people in the order they were added to the search list, so the search list needs to be a queue. Otherwise, you won't get the shortest path.

Debe verificar las personas en el orden en que fueron agregadas a la lista de búsqueda, por lo que la lista de búsqueda debe ser una cola. De lo contrario, no obtendrá el camino más corto.

- Once you check someone, make sure you don't check them again. Otherwise, you might end up in an infinite loop.

Una vez que revises a alguien, asegúrate de no volver a revisarlo. De lo contrario, podrías terminar en un bucle infinito.



# 7 Trees

---

## 7 árboles

---

### In this chapter

### En este capítulo

- You learn what a tree is and the difference between trees and graphs.  
Aprendes qué es un árbol y la diferencia entre árboles y gráficos.
- You get comfortable with running an algorithm over a tree.  
Te sientes cómodo ejecutando un algoritmo sobre un árbol.
- You learn depth-first search and see the difference between depth-first search and breadth-first search.  
Aprenderá la búsqueda en profundidad y verá la diferencia entre la búsqueda en profundidad y la búsqueda en amplitud.
- You learn Huffman coding, a compression algorithm that makes use of trees.  
Aprende codificación Huffman, un algoritmo de compresión que utiliza árboles.

What do compression algorithms and database storage have in common? There is often a tree underneath doing all the hard work. Trees are a subset of graphs. They are worth covering separately as there are many specialized types of trees. For example, Huffman coding, a compression algorithm you will learn in this chapter, uses binary trees.

¿Qué tienen en común los algoritmos de compresión y el almacenamiento de bases de datos? A menudo hay un árbol debajo que hace todo el trabajo duro. Los árboles son un subconjunto de gráficos. Vale la pena cubrirlos por separado, ya que existen muchos tipos especializados de árboles. Por ejemplo, la codificación Huffman, un algoritmo de compresión que aprenderá en este capítulo, utiliza árboles binarios.



Most databases use a balanced tree like a B-tree, which you will learn about in the next chapter. There are so many types of trees out there. These two chapters will give you the vocabulary and concepts you need to understand them.

La mayoría de las bases de datos utilizan un árbol equilibrado como un árbol B, sobre el que aprenderá en el próximo capítulo. Hay tantos tipos de árboles por ahí. Estos dos capítulos le brindarán el vocabulario y los conceptos que necesita para comprenderlos.

## **Your first tree**

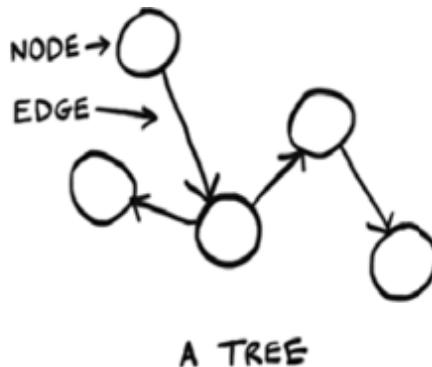
### **Tu primer árbol**

Trees are a type of graph. We will have a more thorough definition later. First, let's learn some terminology and look at an example.

Los árboles son un tipo de gráfico. Tendremos una definición más completa más adelante. Primero, aprendamos algo de terminología y veamos un ejemplo.

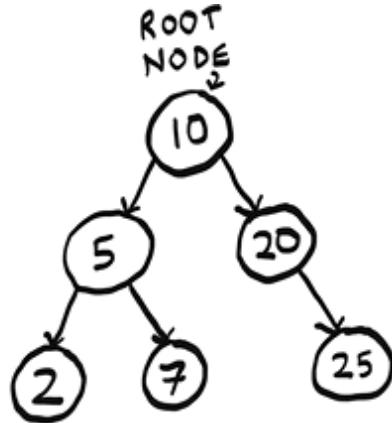
Just like graphs, trees are made of nodes and edges.

Al igual que los gráficos, los árboles están formados por nodos y aristas.



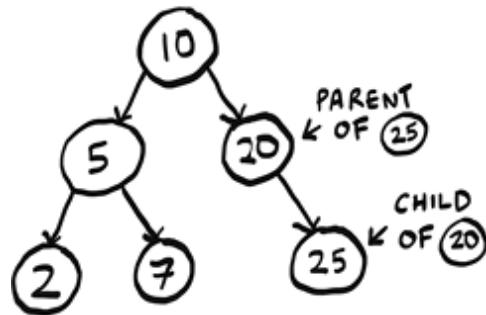
In this book, we will work with rooted trees. Rooted trees have one node that leads to all the other nodes.

En este libro trabajaremos con árboles enraizados. Los árboles enraizados tienen un nodo que conduce a todos los demás nodos.



We will work exclusively with rooted trees, so when I say *tree* in this chapter, I mean a rooted tree. Nodes can have children, and child nodes can have a parent.

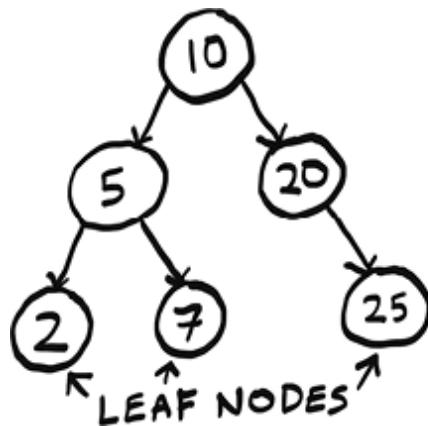
Trabajaremos exclusivamente con árboles enraizados, por lo que cuando digo árbol en este capítulo, me refiero a un árbol enraizado. Los nodos pueden tener hijos y los nodos hijos pueden tener un parente.



In a tree, nodes have at most one parent. The only node with no parents is the root. Nodes with no children are called leaf nodes.

En un árbol, los nodos tienen como máximo un parente. El único nodo que no tiene padres es la raíz. Los nodos sin

hijos se denominan nodos hoja.



If you understand root, leaf, parent, and child, you are ready to read on!

Si comprende raíz, hoja, parente e hijo, iestá listo para seguir leyendo!

## File Directories

### Directorios de archivos

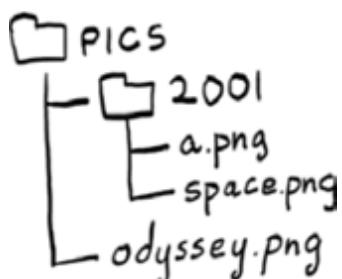
Since a tree is a type of a graph, we can run a graph algorithm on it. In chapter 6, we learned breadth-first search, an algorithm for finding the shortest path in a graph. We are going to use breadth-first search on a tree. If you are not comfortable with breadth-first search, check out chapter 6.

Dado que un árbol es un tipo de gráfico, podemos ejecutar un algoritmo de gráfico en él. En el capítulo 6, aprendimos la búsqueda en amplitud, un algoritmo para encontrar el

camino más corto en un gráfico. Vamos a utilizar la búsqueda en amplitud en un árbol. Si no se siente cómodo con la búsqueda en amplitud, consulte el capítulo 6.

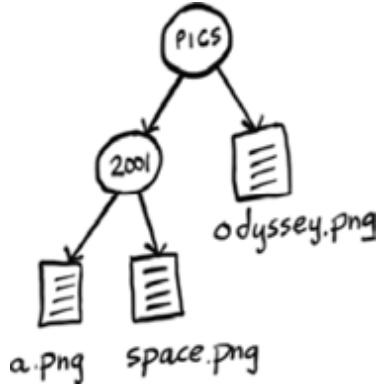
A file directory is a tree that all of us interact with every day. Suppose I have this file directory.

Un directorio de archivos es un árbol con el que todos interactuamos todos los días. Supongamos que tengo este directorio de archivos.



I want to print the name of every file in the pics directory, including all its subdirectories. Here, there is only one subdirectory, 2001. We can use breadth-first search to do this! First, let me show you what this file directory looks like as a tree.

Quiero imprimir el nombre de cada archivo en el directorio de imágenes, incluidos todos sus subdirectorios. Aquí, solo hay un subdirectorio, 2001. ¡Podemos usar la búsqueda en amplitud para hacer esto! Primero, déjame mostrarte cómo se ve este directorio de archivos en forma de árbol.



Since this file directory is a tree, we can run a graph algorithm on it. Earlier, we used breadth-first search as a search algorithm. But search isn't the only thing it's good for. Breadth-first search is a traversal algorithm. That means it is an algorithm that visits every node in a tree—that is, it traverses or walks the tree. That's exactly what we need! We need an algorithm that will go to every file in this tree and print out its name. We will use breadth-first search to list all the files in a directory. The algorithm will also go into subdirectories, find files in there, and print out their names. My logic will be as follows:

Dado que este directorio de archivos es un árbol, podemos ejecutar un algoritmo gráfico en él. Anteriormente, utilizamos la búsqueda en amplitud como algoritmo de búsqueda. Pero la búsqueda no es lo único para lo que sirve. La búsqueda en amplitud es un algoritmo transversal. Eso significa que es un algoritmo que visita cada nodo de un árbol, es decir, atraviesa o camina por el árbol. ¡Eso es exactamente lo que necesitamos! Necesitamos un algoritmo que vaya a cada archivo de este árbol e imprima su nombre. Usaremos la búsqueda en amplitud para enumerar todos los archivos en un directorio. El algoritmo también irá a

subdirectorios, buscará archivos allí e imprimirá sus nombres. Mi lógica será la siguiente:

1. Visit every node in the tree.

Visita cada nodo del árbol.

2. If this node is a file, print out its name.

Si este nodo es un archivo, imprima su nombre.

3. If the node is a folder, add it to a queue of folders to search for files.

Si el nodo es una carpeta, agréguelo a una cola de carpetas para buscar archivos.

The code follows. It is very similar to the mango seller code from chapter 6:

El código sigue. Es muy similar al código de vendedor de mango del capítulo 6:

```
from os import listdir
from os.path import isfile, join
from collections import deque

def printnames(start_dir):
    search_queue = deque()          ①
    search_queue.append(start_dir)
    while search_queue:            ②
        dir = search_queue.popleft()
        for file in sorted(listdir(dir)):
            fullpath = join(dir, file)
            if isfile(fullpath):
                print(file)           ④
            else:
                search_queue.append(fullpath) ⑤

printnames("pics")
```

① We use a queue to keep track of folders to search.

- ① Usamos una cola para realizar un seguimiento de las carpetas para buscar.
- ② While the queue is not empty, pop off a folder to look through.
- ③ Mientras la cola no esté vacía, abra una carpeta para revisarla.
- ④ Loop through every file and folder in this folder.
- ⑤ Recorra todos los archivos y carpetas de esta carpeta.
- ⑥ If it is a file, print out the name.
- ⑦ Si es un archivo, imprima el nombre.
- ⑧ If it is a folder, add it to the queue of folders to search.
- ⑨ Si es una carpeta, agréguela a la cola de carpetas para buscar.

Here, we use a queue like we did in the mango seller example. In the queue we keep track of what folders we still need to search. Of course, in that example, we stopped once we found a mango seller, but here we go through the whole tree.

Aquí usamos una cola como lo hicimos en el ejemplo del vendedor de mangos. En la cola realizamos un seguimiento de las carpetas que aún necesitamos buscar. Por supuesto, en ese ejemplo, nos detuvimos una vez que encontramos un vendedor de mangos, pero aquí recorremos todo el árbol.

There's one other important difference from the mango seller code. Can you spot it?

Hay otra diferencia importante con el código de vendedor de mango. ¿Puedes distinguirlo?

In the mango seller example, remember we had to keep track of whether we had already searched a person:

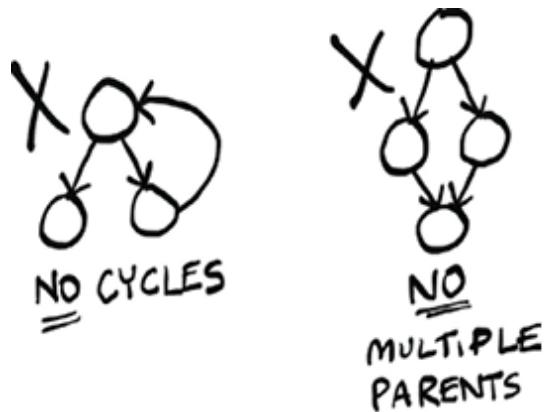
En el ejemplo del vendedor de mangos, recuerde que teníamos que realizar un seguimiento de si ya habíamos

buscado a una persona:

```
...  
    if person not in searched: ①  
        if person_is_seller(person):  
...  
① Only search this person if you haven't already searched them.  
① Busque a esta persona solo si aún no la ha buscado.
```

We don't have to do that here! Trees don't have cycles, and each node only has one parent. There's no way we would accidentally search the same folder more than once or end up in an infinite loop, so there's no need to keep track of which folders we have already searched. There simply isn't a way to revisit a folder.

¡No tenemos que hacer eso aquí! Los árboles no tienen ciclos y cada nodo solo tiene un parent. No hay forma de que busquemos accidentalmente en la misma carpeta más de una vez o terminemos en un bucle infinito, por lo que no es necesario realizar un seguimiento de las carpetas en las que ya hemos buscado. Simplemente no hay forma de volver a visitar una carpeta.



This property of trees has made our code simpler. That's an important takeaway from this chapter: trees don't have cycles.

Esta propiedad de los árboles ha simplificado nuestro código. Ésa es una conclusión importante de este capítulo: los árboles no tienen ciclos.

### A note on symbolic links

### Una nota sobre enlaces simbólicos

You may know what symbolic links are. If you don't, symbolic links are a way to introduce a cycle in a file directory. I could make a symbolic link on macOS or Linux with

Quizás sepas qué son los enlaces simbólicos. Si no lo hace, los enlaces simbólicos son una forma de introducir un ciclo en un directorio de archivos. Podría hacer un enlace simbólico en macOS o Linux con

```
ln -s pics/ pics/2001/pics
```

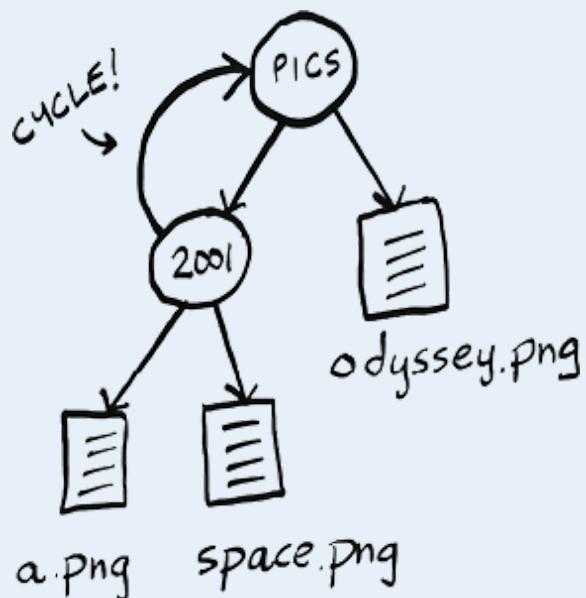
or, on Windows, with

o, en Windows, con

```
mklink /d pics/ pics/2001/pics
```

If I did that, the tree would look like the following.

Si hiciera eso, el árbol se vería como el siguiente.



Now our file directory isn't a tree anymore! To keep things simple, for this example, we are going to ignore symbolic links. If we did have a symbolic link, Python is smart enough to avoid an infinite loop. Here is the error it throws:

¡Ahora nuestro directorio de archivos ya no es un árbol! Para simplificar las cosas, en este ejemplo ignoraremos los enlaces simbólicos. Si tuviéramos un enlace simbólico, Python es lo suficientemente inteligente como para evitar un bucle infinito. Aquí está el error que arroja:

```
OSError: [Errno 62] Too many levels of symbolic links: 'pics/2001/pics'
```

# **A space odyssey: Depth-first search**

## **Una odisea espacial: búsqueda en profundidad**

Let's traverse our file directory again, doing it recursively this time:

Recorramos nuestro directorio de archivos nuevamente, esta vez de forma recursiva:

```
from os import listdir
from os.path import isfile, join

def printnames(dir):
    for file in sorted(listdir(dir)): ①
        fullpath = join(dir, file)
        if isfile(fullpath):
            print(file) ②
        else:
            printnames(fullpath) ③

printnames("pics")
```

- ① Loop through every file and folder in the current folder.
- ① Recorra todos los archivos y carpetas de la carpeta actual.
- ② If it is a file, print out the name.
- ② Si es un archivo, imprima el nombre.
- ③ If it is a folder, call this function recursively on it to look for files and folders.
- ③ Si es una carpeta, llame a esta función de forma recursiva para buscar archivos y carpetas.

Notice that now we are not using a queue. Instead, when we come across a folder, we immediately look inside for more files and folders. Now we have two ways of listing the file

names. But here's the surprising part: *the solutions will print the file names in different orders!*

Observe que ahora no estamos usando una cola. En cambio, cuando nos encontramos con una carpeta, inmediatamente buscamos en su interior más archivos y carpetas. Ahora tenemos dos formas de enumerar los nombres de los archivos. Pero aquí está la parte sorprendente: las soluciones imprimirán los nombres de los archivos en diferentes órdenes!

One prints the names out like this:

Uno imprime los nombres así:

```
a.png  
space.png  
odyssey.png
```

The other prints this:

El otro imprime esto:

```
odyssey.png  
a.png  
space.png
```

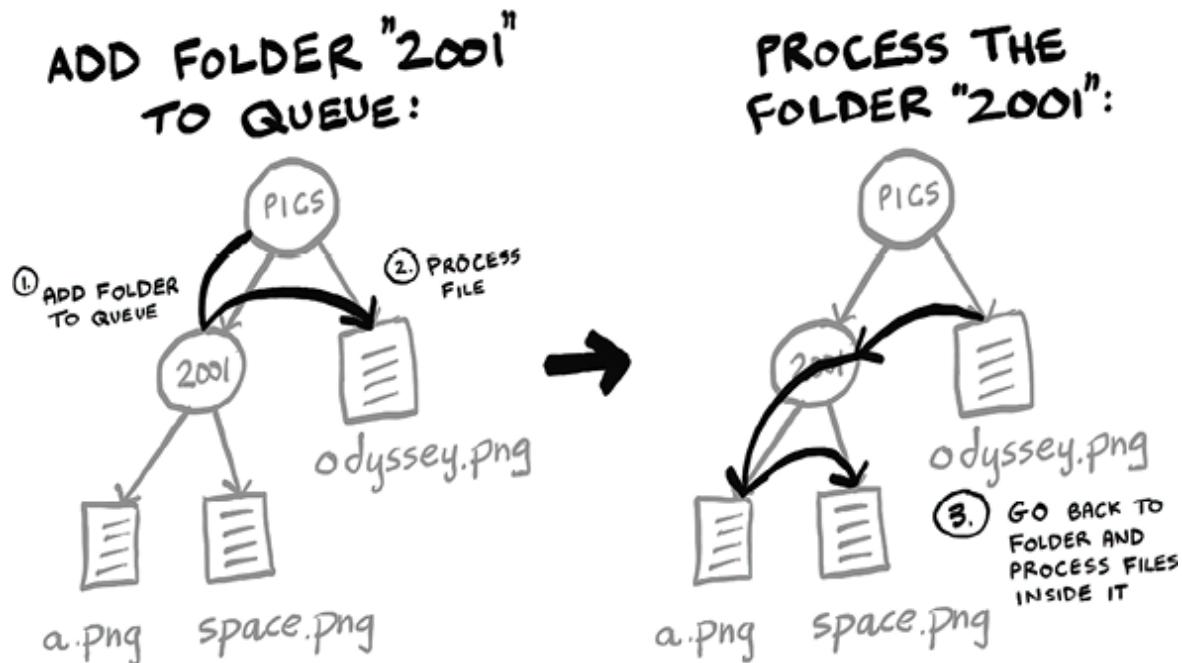
Can you figure out which solution prints which order and why? Try it yourself before moving on.

¿Puedes averiguar qué solución imprime qué orden y por qué? Pruébelo usted mismo antes de continuar.

The first solution uses breadth-first search. When it finds a folder, that folder is added to the queue to be checked later.

So the algorithm goes to the 2001 folder, does not go into it but adds it to the queue to be looked at later, prints all the file names in the pics/ folder, and then goes back to the 2001/ folder and prints the file names in there.

La primera solución utiliza la búsqueda en amplitud. Cuando encuentra una carpeta, esa carpeta se agrega a la cola para verificarla más tarde. Entonces, el algoritmo va a la carpeta 2001, no ingresa en ella, sino que la agrega a la cola para verlo más tarde, imprime todos los nombres de archivos en la carpeta pics/ y luego regresa a la carpeta 2001/ e imprime el archivo. nombres allí.

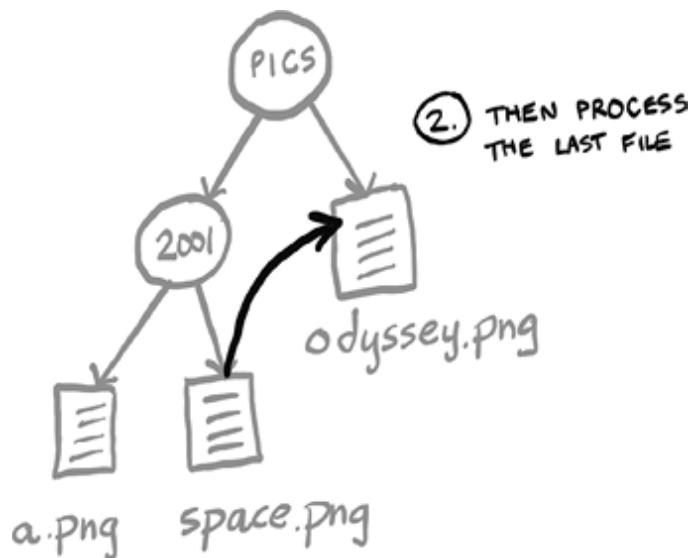
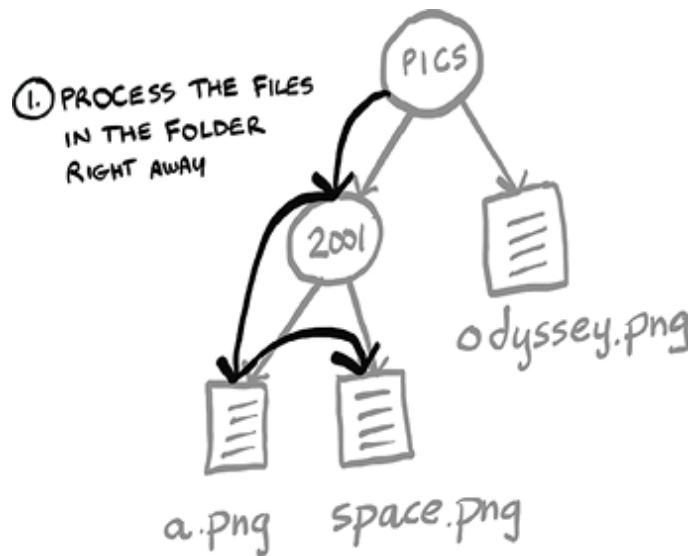


You can see the algorithm visits the 2001 folder first but doesn't look inside. That folder is just added to the queue, and breadth-first search moves on to odyssey.png.

Puede ver que el algoritmo visita primero la carpeta 2001 pero no mira dentro. Esa carpeta simplemente se agrega a la cola y la búsqueda en amplitud pasa a `odyssey.png`.

The second solution uses an algorithm called depth-first search. Depth-first search is also a graph and tree traversal algorithm. When it finds a folder, it looks inside immediately instead of adding it to a queue.

La segunda solución utiliza un algoritmo llamado búsqueda en profundidad. La búsqueda en profundidad también es un algoritmo de recorrido de gráficos y árboles. Cuando encuentra una carpeta, busca dentro inmediatamente en lugar de agregarla a una cola.



The second solution is the one that prints out

La segunda solución es la que imprime.

```
a.png
space.png
odyssey.png
```

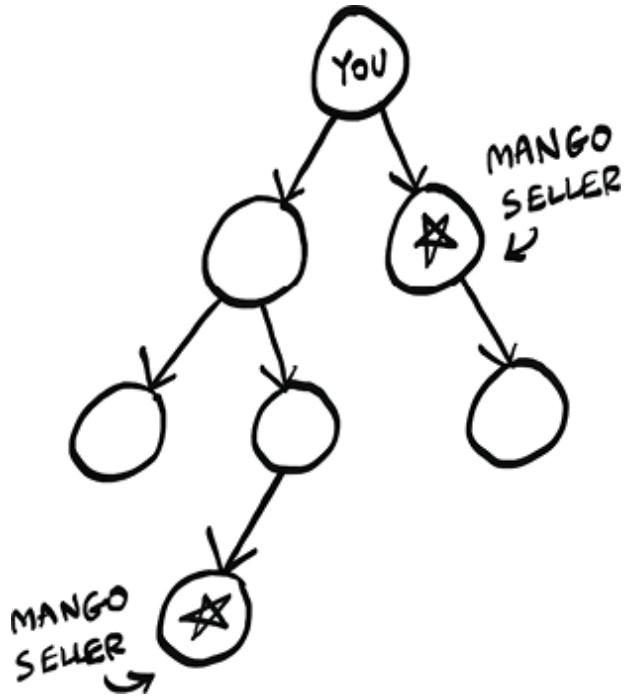
Breadth-first search and depth-first search are closely related, and often where one is mentioned, the other will be also. Both algorithms printed out all the file names, so they

both work for this example. But there is a big difference. Depth-first search cannot be used for finding the shortest path!

La búsqueda en amplitud y la búsqueda en profundidad están estrechamente relacionadas y, a menudo, cuando se menciona una, también se menciona la otra. Ambos algoritmos imprimieron todos los nombres de los archivos, por lo que ambos funcionan para este ejemplo. Pero hay una gran diferencia. ¡La búsqueda en profundidad no se puede utilizar para encontrar el camino más corto!

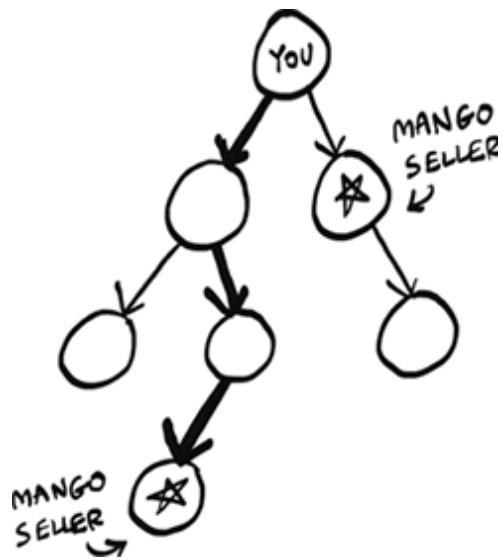
In the mango seller example, we could not have used depth-first search. We rely on the fact that we are checking all our first-degree friends before our second-degree friends, and so on. That's what breadth-first search does. But depth-first search will go as deep as possible right away. It may find you a mango seller three degrees away when you have a closer contact! Suppose the following is your social network.

En el ejemplo del vendedor de mangos, no podríamos haber utilizado la búsqueda en profundidad. Nos basamos en el hecho de que estamos revisando a todos nuestros amigos de primer grado antes que a nuestros amigos de segundo grado, y así sucesivamente. Eso es lo que hace la búsqueda en amplitud. Pero la búsqueda en profundidad será lo más profunda posible de inmediato. ¡Es posible que encuentre un vendedor de mangos a tres grados de distancia si tiene un contacto más cercano! Supongamos que la siguiente es su red social.



Let's say we process nodes in order from left to right. Depth-first search will get to the leftmost child node and go deep.

Digamos que procesamos los nodos en orden de izquierda a derecha. La búsqueda en profundidad llegará al nodo secundario más a la izquierda y profundizará.

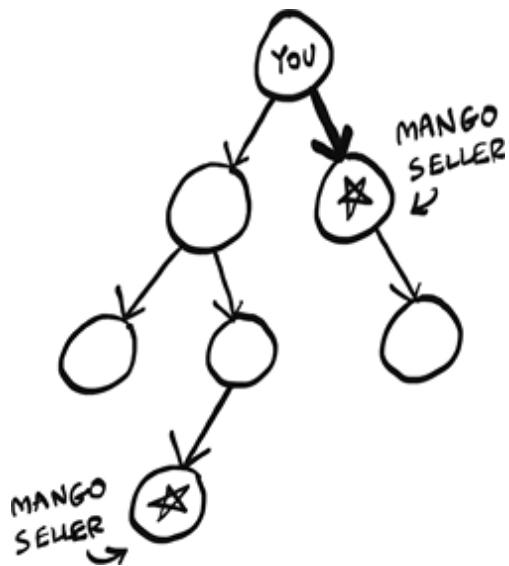


Because the depth-first search went deep on the left node, it failed to realize that the right node is a mango seller that is much closer.

Debido a que la búsqueda en profundidad fue profunda en el nodo izquierdo, no se dio cuenta de que el nodo derecho es un vendedor de mango que está mucho más cerca.

Breadth-first search will correctly find the closest mango seller.

La búsqueda en amplitud encontrará correctamente al vendedor de mango más cercano.



So while both algorithms worked for listing files, only breadth-first search works for finding the shortest path. Depth-first search has other uses. It can be used to find the topological sort, a concept we saw briefly in chapter 6.

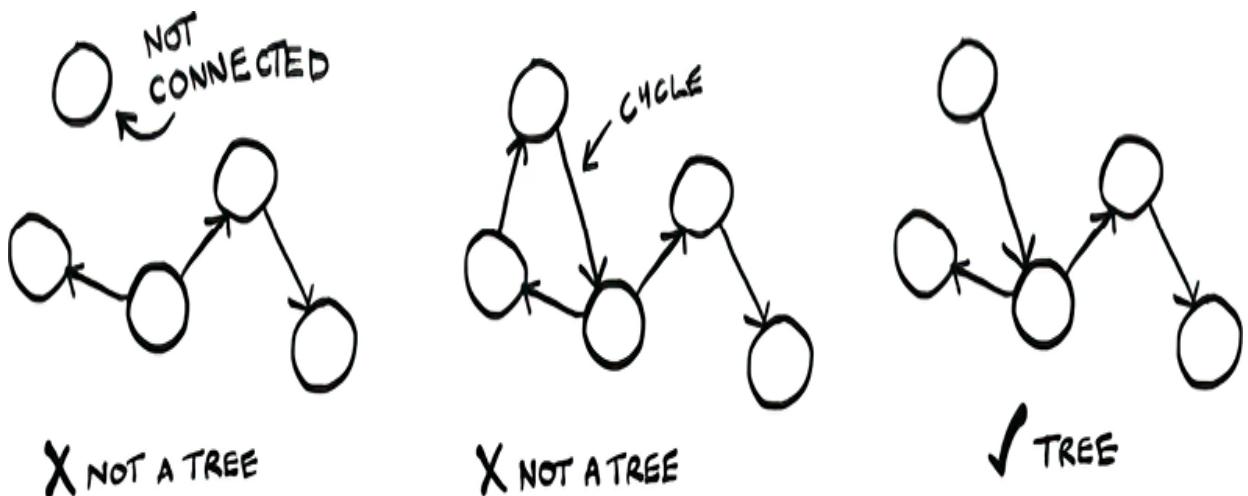
Entonces, si bien ambos algoritmos funcionaron para enumerar archivos, solo la búsqueda en amplitud funciona para encontrar la ruta más corta. La búsqueda en profundidad tiene otros usos. Puede usarse para encontrar el tipo topológico, un concepto que vimos brevemente en el capítulo 6.

## A better definition of trees

### Una mejor definición de los árboles

Now that you have seen an example, it's time for a better definition of a tree. A tree is a *connected, acyclic graph*.

Ahora que ha visto un ejemplo, es hora de definir mejor un árbol. Un árbol es un gráfico acíclico y conectado.



As I had said earlier, we are working exclusively with rooted trees, so our trees all have a root as well. And we are working exclusively with connected graphs. So the most important thing to remember is *trees cannot have cycles*.

Como dije antes, estamos trabajando exclusivamente con árboles enraizados, por lo que todos nuestros árboles también tienen una raíz. Y estamos trabajando exclusivamente con gráficos conectados. Entonces, lo más importante que debemos recordar es que los árboles no pueden tener ciclos.

Now we have seen a tree in action, let's zoom in on one specific type of tree.

Ahora que hemos visto un árbol en acción, acerquémonos a un tipo específico de árbol.

## **Binary trees**

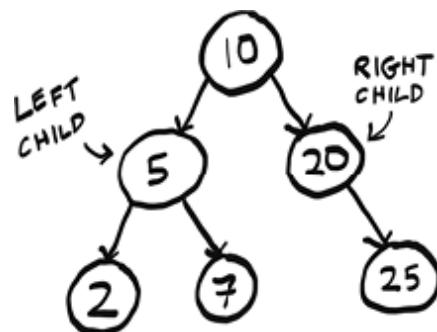
### **árboles binarios**

Computer science is full of different types of trees. Binary trees are a very common type of tree. For the rest of this chapter and most of the next, we will work with binary trees.

La informática está llena de diferentes tipos de árboles. Los árboles binarios son un tipo de árbol muy común. Durante el resto de este capítulo y la mayor parte del siguiente, trabajaremos con árboles binarios.

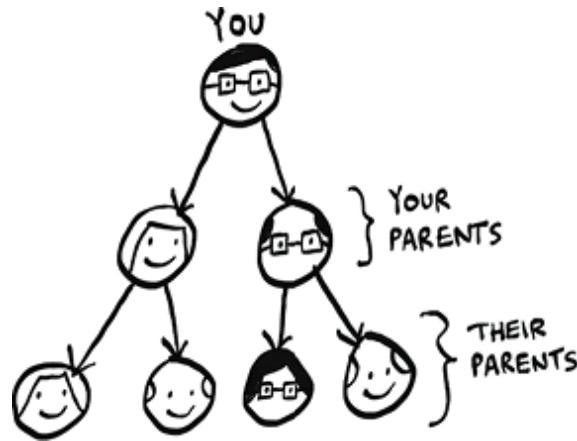
A binary tree is a special type of tree where nodes can have at most two children (hence the name *binary*, meaning *two*). These are traditionally called left child and right child.

Un árbol binario es un tipo especial de árbol donde los nodos pueden tener como máximo dos hijos (de ahí el nombre binario, que significa dos). Estos se denominan tradicionalmente hijo izquierdo e hijo derecho.



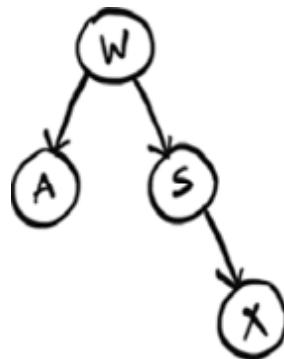
An ancestry tree is an example of a binary tree since everyone has two biological parents.

Un árbol de ascendencia es un ejemplo de árbol binario, ya que todo el mundo tiene dos padres biológicos.



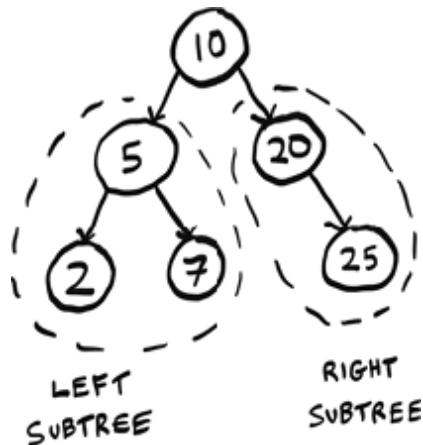
In that example, there's a clear connection between nodes—they are all family. However, the data can be totally arbitrary.

En ese ejemplo, hay una conexión clara entre los nodos: todos son familia. Sin embargo, los datos pueden ser totalmente arbitrarios.



The important thing is you never have more than two children. Sometimes people refer to the left subtree or right subtree.

Lo importante es que nunca tengas más de dos hijos. A veces la gente se refiere al subárbol izquierdo o al subárbol derecho.



Binary trees show up everywhere in computer science. We are going to spend the rest of this chapter looking at an example that uses a binary tree.

Los árboles binarios aparecen en todas partes en la informática. Pasaremos el resto de este capítulo analizando un ejemplo que utiliza un árbol binario.

## Huffman coding

### Codificación Huffman

Huffman coding is a neat example of using binary trees. It's also the foundation for text compression algorithms. We won't describe the algorithm but will spend time focusing on how it works and how it makes clever use of trees.

La codificación de Huffman es un buen ejemplo del uso de árboles binarios. También es la base de los algoritmos de compresión de texto. No describiremos el algoritmo, pero dedicaremos tiempo a centrarnos en cómo funciona y cómo hace un uso inteligente de los árboles.

First, a little background. To know how compression works, we need to know how much space a text file takes. Suppose we have a text file with just one word: *tilt*. How much space does that use? You can use the `stat` command (available on Unix). First, save the word in a file called `test.txt`. Then, using `stat`,

Primero, un poco de historia. Para saber cómo funciona la compresión, necesitamos saber cuánto espacio ocupa un archivo de texto. Supongamos que tenemos un archivo de texto con una sola palabra: inclinación. ¿Cuánto espacio usa eso? Puede utilizar el comando `stat` (disponible en Unix). Primero, guarde la palabra en un archivo llamado `test.txt`. Luego, usando `stat`,

```
$ cat test.txt  
tilt  
  
$ stat -f%z test.txt  
4
```

so that file takes up 4 bytes: 1 byte per character.

entonces ese archivo ocupa 4 bytes: 1 byte por carácter.

This makes sense. Assuming we are using ISO-8859-1 (see the following sidebar for what this means), each letter takes

up exactly 1 byte. For example, the letter *a* is ISO-8859-1 code 97, which I can write in binary as 01100001. That is 8 bits. A bit is a digit that can be either 0 or 1. And there are eight of them. Eight bits is 1 byte. So the letter *a* is represented using 1 byte. ISO-8859-1 code goes from 00000000, which represents the null character, all the way to 11111111, which represents ÿ (Latin lowercase letter *y* with diaeresis). There are 256 possible combinations of 0s and 1s with 8 bits, so the ISO-8859-1 code allows for 256 possible letters.

Esto tiene sentido. Suponiendo que estamos utilizando ISO-8859-1 (consulte la siguiente barra lateral para saber qué significa esto), cada letra ocupa exactamente 1 byte. Por ejemplo, la letra *a* es el código ISO-8859-1 97, que puedo escribir en binario como 01100001. Es decir, 8 bits. Un bit es un dígito que puede ser 0 o 1. Y hay ocho. Ocho bits son 1 byte. Entonces la letra *a* se representa usando 1 byte. El código ISO-8859-1 va desde 0000, que representa el carácter nulo, hasta 1111, que representa ÿ (letra *y* minúscula latina con diéresis). Hay 256 combinaciones posibles de 0 y 1 con 8 bits, por lo que el código ISO-8859-1 permite 256 letras posibles.

## Character encoding

## Codificación de caracteres

As this example will show you, there are many different ways to encode characters. That is, the letter *a* could be written in binary in many different ways.

Como le mostrará este ejemplo, existen muchas formas diferentes de codificar caracteres. Es decir, la letra a podría escribirse en binario de muchas formas distintas.

It started with ASCII. In the 1960s, ASCII was created. ASCII is a 7-bit encoding. Unfortunately, ASCII did not include a lot of characters. ASCII does not include any characters with umlauts (ü or ö, for example) or common currencies like the British pound or Japanese yen.

Comenzó con ASCII. En la década de 1960 se creó ASCII. ASCII es una codificación de 7 bits. Lamentablemente, ASCII no incluía muchos caracteres. ASCII no incluye caracteres con diéresis (ü o ö, por ejemplo) ni monedas comunes como la libra esterlina o el yen japonés.

So ISO-8859-1 was created. ISO-8859-1 is an 8-bit encoding, so it doubles the number of characters that ASCII provided. We went from 128 characters to 256 characters. But this was still not enough, and countries began making their own encodings. For example, Japan has several encodings for Japanese since ISO-8859-1 and ASCII were focused on European languages. The whole situation was a mess until Unicode was introduced.

Entonces se creó ISO-8859-1. ISO-8859-1 es una codificación de 8 bits, por lo que duplica la cantidad de caracteres que proporcionó ASCII. Pasamos de 128 caracteres a 256 caracteres. Pero esto todavía no fue suficiente y los países comenzaron a crear sus propias codificaciones. Por ejemplo, Japón tiene varias codificaciones para japonés ya que ISO-8859-1 y ASCII se centraron en idiomas europeos. Toda la situación fue un desastre hasta que se introdujo Unicode.

Unicode is an encoding standard. It aims to provide characters for any language. Unicode has 149,186 characters as of version 15—quite a jump from 256! More than 1,000 of these are emojis.

Unicode es un estándar de codificación. Su objetivo es proporcionar caracteres para cualquier idioma. Unicode tiene 149,186 caracteres a partir de la versión 15, ¡un gran salto desde 256! Más de 1.000 de ellos son emojis.

Unicode is the standard, but you need to use an encoding that follows the standard. The most popular encoding today is UTF-8. UTF-8 is variable-length character encoding, which means characters can be anywhere from 1 to 4 bytes (8–32 bits).

Unicode es el estándar, pero es necesario utilizar una codificación que siga el estándar. La codificación más popular en la actualidad es UTF-8. UTF-8 es una codificación de caracteres de longitud variable, lo que significa que los caracteres pueden tener entre 1 y 4 bytes (8 a 32 bits).

You don't need to worry too much about this. I've kept the example simple intentionally by using ISO-8859-1, which is 8 bits—a nice consistent quantity of bits to work with.

No necesitas preocuparte demasiado por esto. He mantenido el ejemplo simple intencionalmente usando ISO-8859-1, que es de 8 bits, una buena cantidad consistente de bits con los que trabajar.

Just remember these takeaways:

Solo recuerda estas conclusiones:

- Compression algorithms try to reduce the number of bits needed to store each character.  
Los algoritmos de compresión intentan reducir la cantidad de bits necesarios para almacenar cada carácter.
- If you need to pick an encoding for a project, UTF-8 is a good default choice.  
Si necesita elegir una codificación para un proyecto, UTF-8 es una buena opción predeterminada.

Let's decode some binary to ISO-8859-1 together:

011100100110000101100100. You can Google an ISO-8859-1 table or a binary-to-ISO-8859-1 converter to make this easier.

Decodifiquemos juntos algo de binario a ISO-8859-1:  
011100100110000101100100. Puede buscar en Google una tabla ISO-8859-1 o un convertidor de binario a ISO-8859-1 para hacerlo más fácil.

First, we know that each letter is 8 bits, so I am going to divide this into chunks of 8 bits to make it easier to read:

Primero, sabemos que cada letra tiene 8 bits, así que voy a dividir esto en trozos de 8 bits para que sea más fácil de leer:

```
01110010 01100001 01100100
```

Great, now we see that there are three letters. Looking them up in an ISO-8859-1 table, I see they spell out `rad`: 01110010 is `r`, and so on. This is how your text editor takes the binary data in a text file and displays it as ISO-8859-1. You can view the binary information by using `xxd`. This utility is available on Unix. Here is how `tilt` looks in binary:

Genial, ahora vemos que hay tres letras. Al buscarlos en una tabla ISO-8859-1, veo que escriben `rad`: 01110010 es `r`, y así sucesivamente. Así es como su editor de texto toma los datos binarios en un archivo de texto y los muestra como ISO-8859-1. Puede ver la información binaria usando `xxd`. Esta utilidad está disponible en Unix. Así es como se ve `tilt` en binario:

```
$ xxd -b test.txt
00000000: 01110100 01101001 01101100 01110100
tilt
```

Here is where the compression comes in. For the word `tilt`, we don't need 256 possible letters; we just need three. So we don't need 8 bits; we only need 2. We could come up with our own 2-bit code just for these three letters:

Aquí es donde entra en juego la compresión. Para la palabra inclinación, no necesitamos 256 letras posibles; sólo necesitamos tres. Entonces no necesitamos 8 bits; sólo necesitamos 2. Podríamos crear nuestro propio código de 2 bits solo para estas tres letras:

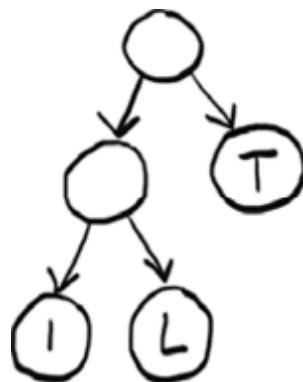
```
t = 00
i = 01
l = 10
```

Here is how we could write *tilt* using our new code: 00011000. I can make this easier to read by adding spaces again: 00 01 10 00. If you compare it to the mapping, you'll see this spells out *tilt*.

Así es como podríamos escribir inclinación usando nuestro nuevo código: 00011000. Puedo hacer que esto sea más fácil de leer agregando espacios nuevamente: 00 01 10 00. Si lo comparas con el mapeo, verás que esto explica la inclinación.

This is what Huffman coding does: it looks at the characters being used and tries to use less than 8 bits. That is how it compresses the data. Huffman coding generates a tree.

Esto es lo que hace la codificación Huffman: analiza los caracteres que se utilizan e intenta utilizar menos de 8 bits. Así es como comprime los datos. La codificación de Huffman genera un árbol.



You can use this tree to find the code for each letter. Starting at the root node, find a path down to the letter *L*. Whenever you choose a left branch, append a 0 to your code. When

you choose a right branch, append 1. When you get to a letter, stop progressing down the tree. So the code for the letter *L* is 01. Here are the three codes given by the tree:

Puede utilizar este árbol para encontrar el código de cada letra. Comenzando en el nodo raíz, busque una ruta hasta la letra L. Siempre que elija una rama izquierda, agregue un 0 a su código. Cuando elijas una rama derecha, agrega 1. Cuando llegues a una letra, deja de avanzar por el árbol. Entonces el código para la letra L es 01. Aquí están los tres códigos dados por el árbol:

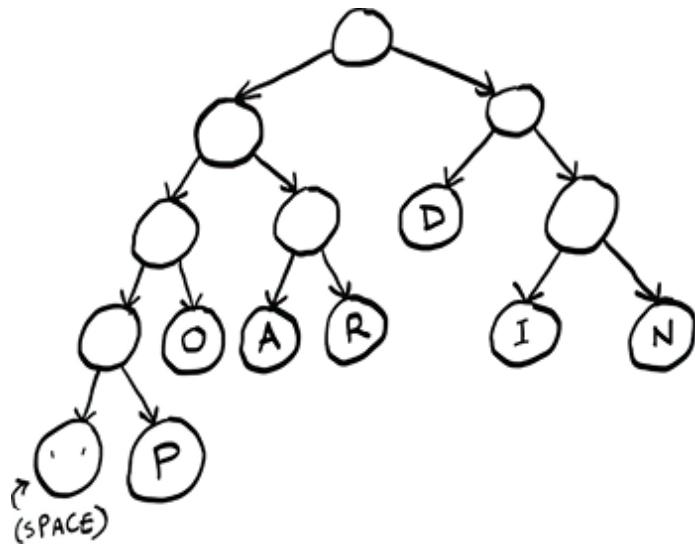
```
i = 00  
l = 01  
t = 1
```

Notice that the letter T has a code of just one digit. Unlike ISO-8859-1, *in Huffman coding, the codes don't all have to be the same length*. This is important. Let's see another example to understand why.

Observe que la letra T tiene un código de solo un dígito. A diferencia de ISO-8859-1, en la codificación Huffman, no es necesario que todos los códigos tengan la misma longitud. Esto es importante. Veamos otro ejemplo para entender por qué.

Now we want to compress the phrase "paranoid android." Here is the tree generated by the Huffman coding algorithm.

Ahora queremos comprimir la frase "android paranoico". Aquí está el árbol generado por el algoritmo de codificación de Huffman.



Check yourself: What is the code for the letter *P*? Try it yourself before reading on. It is 0001. What about the letter *D*? It is 10.

Compruébalo tú mismo: ¿Cuál es el código de la letra *P*? Pruébelo usted mismo antes de seguir leyendo. Es 0001. ¿Qué pasa con la letra *D*? Son 10.



In this case, there are actually three different possible lengths! Suppose we try to decode some binary data: 01101010. We see the problem right away: we can't chunk this up the way we did with ISO-8859-1! While all ISO-8859-1 codes were eight digits, here the code could be two,

three, or four digits. *Since the code length varies, we can't use chunking anymore.*

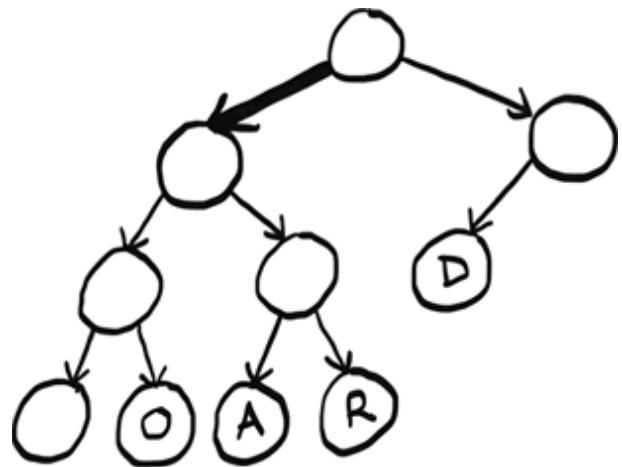
En este caso, en realidad hay tres longitudes posibles diferentes! Supongamos que intentamos decodificar algunos datos binarios: 01101010. Vemos el problema de inmediato: no podemos dividir esto como lo hicimos con ISO-8859-1! Si bien todos los códigos ISO-8859-1 tenían ocho dígitos, aquí el código podía tener dos, tres o cuatro dígitos. Dado que la longitud del código varía, ya no podemos utilizar la fragmentación.

Instead, we need to look at one digit at a time, as if we are looking at a tape.

En cambio, debemos mirar un dígito a la vez, como si estuviéramos mirando una cinta.

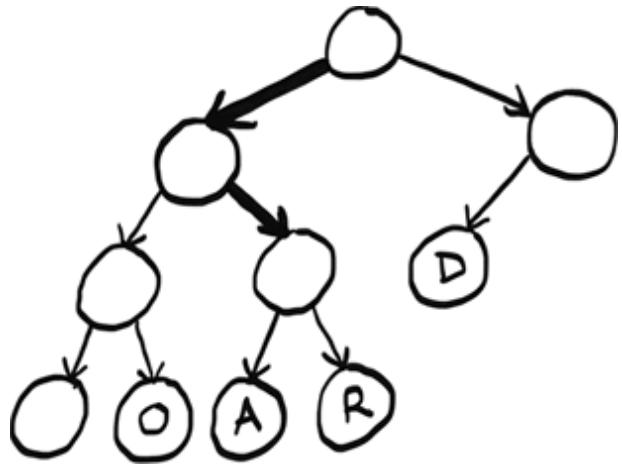
Here's how to do it: first number is 0, so go left (I'm only showing part of the tree here).

Así es como se hace: el primer número es 0, así que ve a la izquierda (aquí solo muestro parte del árbol).



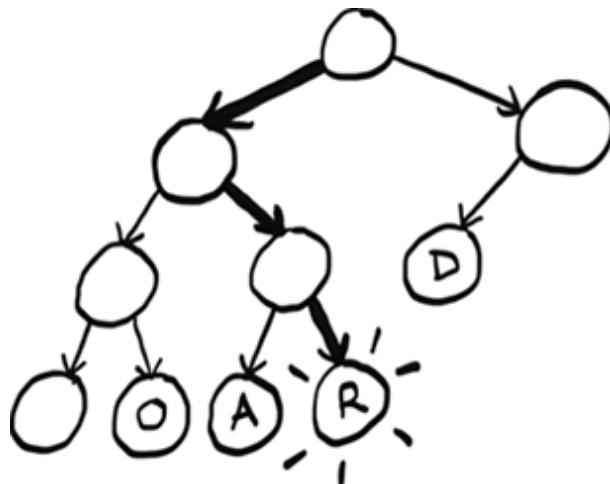
Then we get a 1, so we go right.

Luego obtenemos un 1, así que vamos a la derecha.



Then we get another 1, so we go right again.

Luego obtenemos otro 1, así que volvemos a la derecha.



Aha! We found a letter. This is the binary data we have left: 01010. We can start over at the root node and find the other letters. Try decoding the rest yourself and then read on. Did you get the word? It was *rad*. This is a big difference between Huffman coding and ISO-8859-1. The codes can vary, so the decoding needs to be done differently.

Ajá! Encontramos una carta. Estos son los datos binarios que nos quedan: 01010. Podemos comenzar de nuevo en el nodo raíz y encontrar las otras letras. Intente decodificar el resto usted mismo y luego siga leyendo. ¿Recibiste la palabra? Fue genial. Ésta es una gran diferencia entre la codificación Huffman y la ISO-8859-1. Los códigos pueden variar, por lo que la decodificación debe realizarse de manera diferente.

It is more work to do it this way instead of chunking. But there is one big benefit. Notice that the letters that show up more often have shorter codes. *D* appears three times, so its code is just two digits versus *I*, which appears twice, and *P*, which appears only once. Instead of assigning 4 bits to

everything, we can compress frequently used letters even more. You can see how, in a longer piece of text, this would be a big savings!

Es más trabajo hacerlo de esta manera en lugar de fragmentarlo. Pero hay un gran beneficio. Observe que las letras que aparecen con más frecuencia tienen códigos más cortos. D aparece tres veces, por lo que su código tiene solo dos dígitos versus I, que aparece dos veces, y P, que aparece solo una vez. En lugar de asignar 4 bits a todo, podemos comprimir aún más las letras de uso frecuente. Puedes ver cómo, en un texto más largo, iesto supondría un gran ahorro!

Now that we understand at a high level how Huffman coding works, let's see what properties of trees Huffman is taking advantage of here.

Ahora que entendemos a un alto nivel cómo funciona la codificación de Huffman, veamos qué propiedades de los árboles aprovecha Huffman aquí.

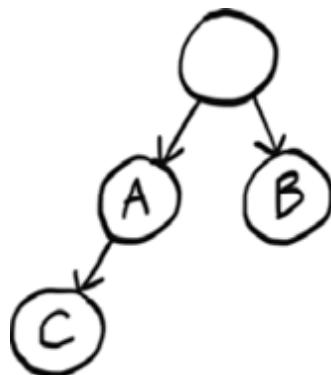
First, could there be overlap between codes? Take this code for example:

En primer lugar, ¿podría haber superposición entre códigos? Tome este código por ejemplo:

```
a = 0  
b = 1  
c = 00
```

Now if you see the binary 001, is that *AAB* or *CB*? *c* and *a* share part of their code, so it's unclear. Here is what the tree for this code would look like.

Ahora bien, si ve el binario 001, ¿es *AAB* o *CB*? *c* y comparten parte de su código, por lo que no está claro. Así es como se vería el árbol de este código.



We pass *A* on the way to *C*, which causes the problem.

Pasamos por *A* de camino a *C*, lo que causa el problema.

That's not a problem with Huffman coding because letters only show up at leaf nodes. And there's a unique path from the root to each leaf node—that's one of the properties of trees. So we can guarantee overlap is not a problem.

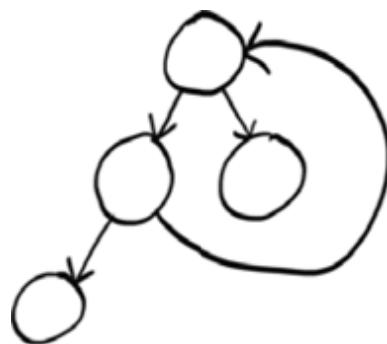
Eso no es un problema con la codificación de Huffman porque las letras sólo aparecen en los nodos de las hojas. Y hay un camino único desde la raíz hasta cada nodo de la hoja: esa es una de las propiedades de los árboles. Así que podemos garantizar que la superposición no sea un problema.

This also guarantees there is only one code for each letter. Having multiple paths to each letter would mean there are multiple codes assigned to each letter, which would be unnecessary.

Esto también garantiza que solo haya un código para cada letra. Tener múltiples rutas a cada letra significaría que hay múltiples códigos asignados a cada letra, lo cual sería innecesario.

When we read the code one digit at a time, we are assuming we will eventually end up at a letter. If this was a graph with a cycle, we couldn't make that assumption. We could get stuck in the cycle and end up in an infinite loop.

Cuando leemos el código un dígito a la vez, asumimos que eventualmente terminaremos en una letra. Si se tratara de un gráfico con un ciclo, no podríamos hacer esa suposición. Podríamos quedarnos atrapados en el ciclo y terminar en un bucle infinito.



But since this is a tree, we know there are no cycles, so we are guaranteed to end up at some letter.

Pero como se trata de un árbol, sabemos que no hay ciclos, por lo que tenemos la garantía de terminar en alguna letra.

We are using a rooted tree. Rooted trees have a root node, which is important because we need to know where to start! Graphs do not necessarily have a root node.

Estamos usando un árbol enraizado. Los árboles enraizados tienen un nodo raíz, lo cual es importante porque necesitamos saber por dónde empezar. Los gráficos no necesariamente tienen un nodo raíz.

Finally, the type of tree used here is called a *binary tree*. Binary trees can have at most two children—the left child and the right child. This makes sense because binary only has two digits. If there was a third child, it would be unclear what digit it is supposed to represent.

Finalmente, el tipo de árbol utilizado aquí se denomina árbol binario. Los árboles binarios pueden tener como máximo dos hijos: el hijo izquierdo y el hijo derecho. Esto tiene sentido porque el binario solo tiene dos dígitos. Si hubiera un tercer hijo, no estaría claro qué dígito se supone que representa.

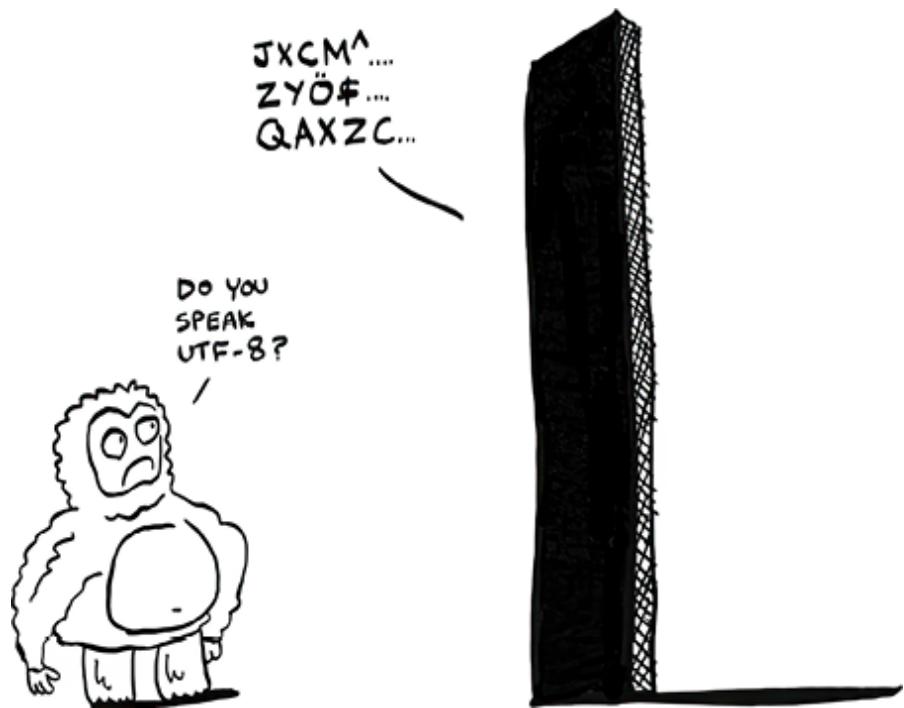
This chapter introduced you to trees. In the next chapter, we will see some different types of trees and what they are used for.

Este capítulo le presentó los árboles. En el próximo capítulo, veremos algunos tipos diferentes de árboles y para qué se utilizan.

## **Recap**

### **Resumen**

- Trees are a type of graph, but trees don't have cycles.  
Los árboles son un tipo de gráfico, pero los árboles no tienen ciclos.
- Depth-first search is another graph traversal algorithm.  
It can't be used to find shortest paths.  
La búsqueda en profundidad es otro algoritmo de recorrido de gráficos. No se puede utilizar para encontrar caminos más cortos.
- A binary tree is a special type of tree where nodes can have, at most, two children.  
Un árbol binario es un tipo especial de árbol donde los nodos pueden tener, como máximo, dos hijos.
- There are many different types of character encodings.  
Unicode is the international standard, and UTF-8 is the most common Unicode encoding.  
Hay muchos tipos diferentes de codificaciones de caracteres. Unicode es el estándar internacional y UTF-8 es la codificación Unicode más común.



# **8 Balanced trees**

---

## **8 árboles equilibrados**

---

### **In this chapter**

#### **En este capítulo**

- You learn about a new data structure called binary search trees (BSTs).

Aprenderá sobre una nueva estructura de datos llamada árboles de búsqueda binarios (BST).

- You learn about balanced trees and why they often perform better than arrays or linked lists.

Aprenderá sobre los árboles equilibrados y por qué a menudo funcionan mejor que las matrices o las listas vinculadas.

- You also learn about AVL trees, a type of balanced BST. In the worst-case scenario, binary trees can be slow. A balanced tree will help them perform effectively.

También aprenderá sobre los árboles AVL, un tipo de BST equilibrado. En el peor de los casos, los árboles binarios pueden ser lentos. Un árbol equilibrado les ayudará a desempeñarse con eficacia.

In the last chapter, you learned about the new data structure, trees. Now that you and trees are best friends, it's time to see what they are used for. When arrays and linked lists fail to deliver the desired performance, a good next step is to try a tree. In this chapter, we'll discuss the performance

that trees can offer. We'll then explore a special type of tree that can offer exceptional performance, called a balanced tree.

En el último capítulo, aprendiste sobre la nueva estructura de datos, los árboles. Ahora que usted y los árboles son mejores amigos, es hora de ver para qué se utilizan. Cuando las matrices y las listas enlazadas no logran ofrecer el rendimiento deseado, un buen paso siguiente es probar un árbol. En este capítulo, analizaremos el rendimiento que pueden ofrecer los árboles. Luego exploraremos un tipo especial de árbol que puede ofrecer un rendimiento excepcional, llamado árbol equilibrado.



## ***A balancing act***

### ***Un acto de equilibrio***

Remember binary search from way back in chapter 1? Using binary search, we are able to find information much more quickly than if we did a simple search using  $O(\log n)$  instead of  $O(n)$ . There is one problem, though: insertion. Sure, searching takes  $O(\log n)$  time, but the array needs to be sorted. If you want to insert a new number into your sorted array, it will take  $O(n)$  time. The issue is making a spot for the new value. You need to move a bunch of values to make room.

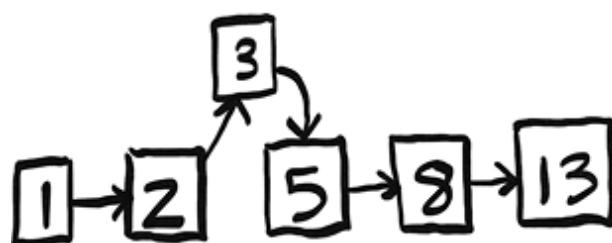
¿Recuerdas la búsqueda binaria del capítulo 1? Usando la búsqueda binaria, podemos encontrar información mucho más rápidamente que si hiciéramos una búsqueda simple usando  $O(\log n)$  en lugar de  $O(n)$ . Sin embargo, hay un problema: la inserción. Claro, la búsqueda lleva  $O(\log n)$

tiempo, pero es necesario ordenar la matriz. Si desea insertar un nuevo número en su matriz ordenada, tomará  $O(n)$  tiempo. La cuestión es hacerle un hueco al nuevo valor. Necesitas mover un montón de valores para hacer espacio.



If only we could insert like we do in a linked list, where we just need to change a couple of pointers.

Si tan solo pudiéramos insertar como lo hacemos en una lista vinculada, donde solo necesitamos cambiar un par de punteros.



But searching is linear time in linked lists. How can we get the best of both worlds?

Pero la búsqueda es un tiempo lineal en listas enlazadas.  
¿Cómo podemos obtener lo mejor de ambos mundos?

## **Improving insertion speed with trees**

### **Mejorando la velocidad de inserción con árboles**

So we really want the search speed of a sorted array with a faster insertion speed. We know that insertions are faster in linked lists. So we want some kind of data structure that combines these ideas.

Entonces realmente queremos la velocidad de búsqueda de una matriz ordenada con una velocidad de inserción más rápida. Sabemos que las inserciones son más rápidas en listas enlazadas. Por eso queremos algún tipo de estructura de datos que combine estas ideas.

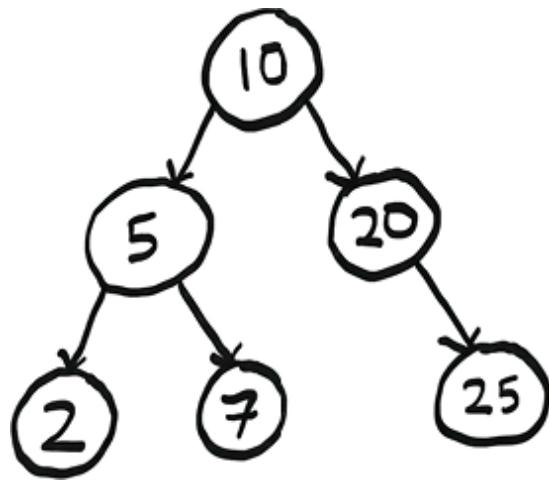
	SEARCH	INSERT
SORTED ARRAY	$O(\log n)$	$O(n)$
LINKED LIST	$O(n)$	$O(1)$
???	$O(\log n)$	FASTER THAN $O(n)$

And that structure is a tree! There are dozens of different types of trees to choose from, so I specifically mean a balanced binary search tree (BST). In this chapter, we will see how a BST works, and then we will learn how to balance it.

¡Y esa estructura es un árbol! Hay docenas de tipos diferentes de árboles para elegir, por lo que me refiero específicamente a un árbol de búsqueda binario equilibrado (BST). En este capítulo, veremos cómo funciona un BST y luego aprenderemos cómo equilibrarlo.

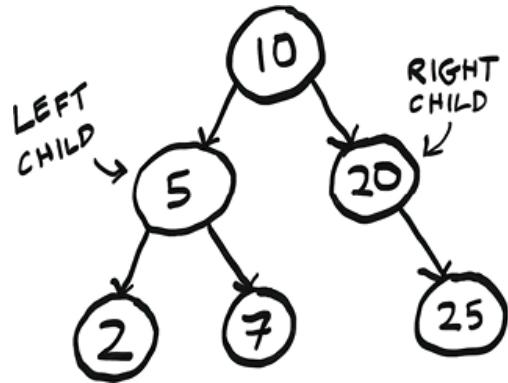
BSTs are a type of binary tree. Here is an example of a BST.

Los BST son un tipo de árbol binario. A continuación se muestra un ejemplo de BST.



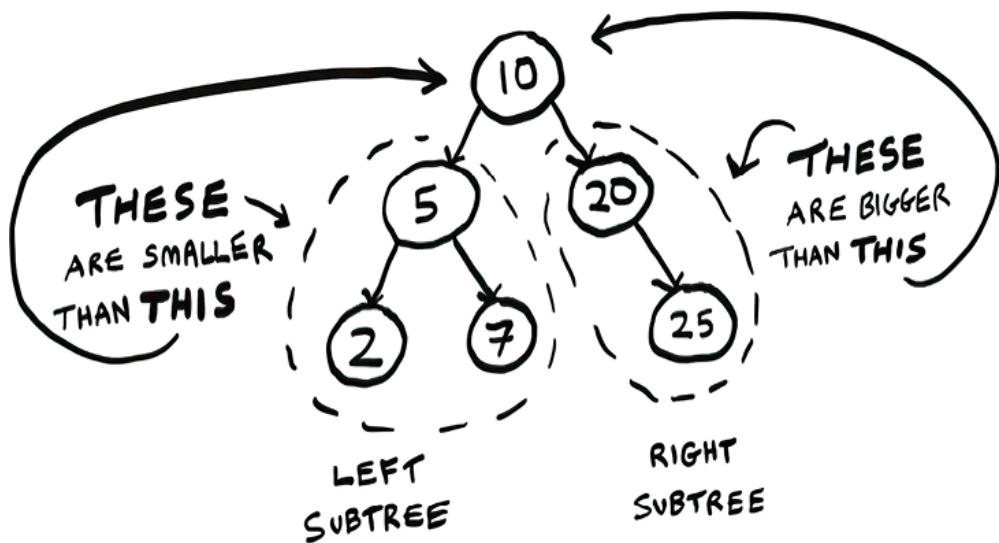
Just like a binary tree, each node has up to two children: the left child and the right child. But it has a special property that makes it a BST: the value of the left child is *always smaller* than the node, and the value of the right child is *always greater*. So for node 10, its left child has a smaller value (5), and its right child has a bigger value (20).

Al igual que un árbol binario, cada nodo tiene hasta dos hijos: el hijo izquierdo y el hijo derecho. Pero tiene una propiedad especial que lo convierte en un BST: el valor del hijo izquierdo siempre es menor que el nodo y el valor del hijo derecho siempre es mayor. Entonces, para el nodo 10, su hijo izquierdo tiene un valor menor (5) y su hijo derecho tiene un valor mayor (20).



Not only that, all the numbers in the left child subtree are smaller than the node!

No solo eso, todos los números en el subárbol secundario izquierdo son más pequeños que el nodo!

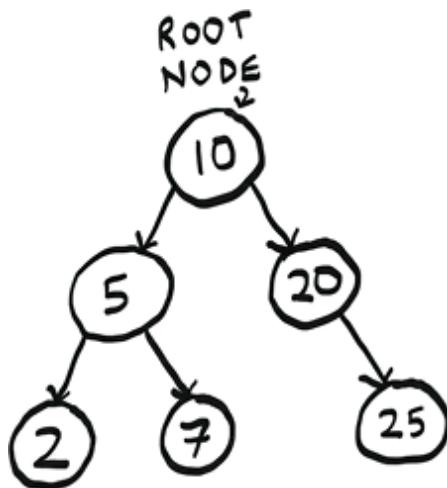


This special property means searches will be very fast.

Esta propiedad especial significa que las búsquedas serán muy rápidas.

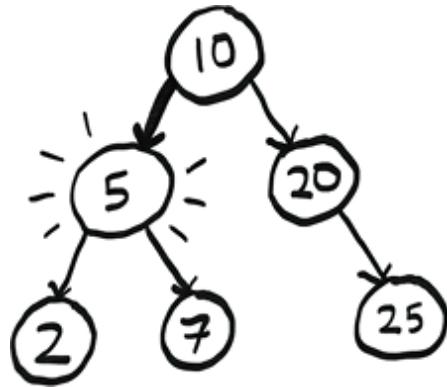
Let's see if the number 7 is in this tree. Here is how we do it. Start at the root node.

Veamos si el número 7 está en este árbol. Así es como lo hacemos. Comience en el nodo raíz.



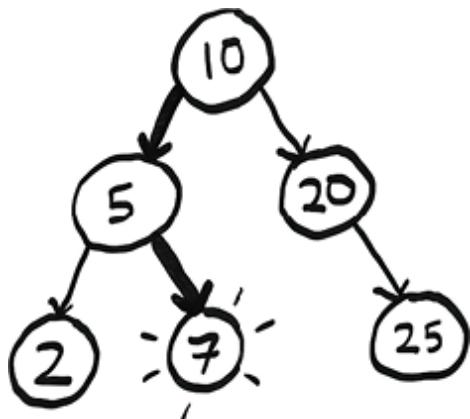
Seven is smaller than 10, so we check the left subtree. Remember, all the nodes with smaller values are on the left, and all the nodes with bigger values are on the right. So we know right away we don't need to check the nodes on the right since 7 won't be there. If we go left from the 10, we get to the 5.

Siete es menor que 10, por lo que verificamos el subárbol izquierdo. Recuerde, todos los nodos con valores más pequeños están a la izquierda y todos los nodos con valores más grandes están a la derecha. Entonces sabemos de inmediato que no necesitamos verificar los nodos de la derecha ya que 7 no estará allí. Si vamos a la izquierda desde el 10, llegamos al 5.



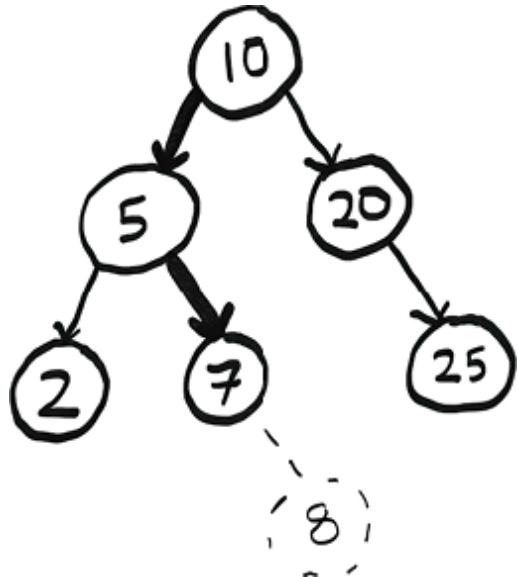
Seven is bigger than 5, so let's go right this time.

Siete es mayor que 5, así que esta vez vayamos bien.



We found it! Now let's look for another number, 8. We follow the exact same path.

iLo encontramos! Ahora busquemos otro número, el 8. Seguimos exactamente el mismo camino.



Except now it is not there! If it was in the tree, it would be right where the dotted node is. The whole reason we are talking about trees is to see if they are faster than arrays and linked lists. So let's talk about the performance of this tree. To do that, we need to look at the height of the tree.

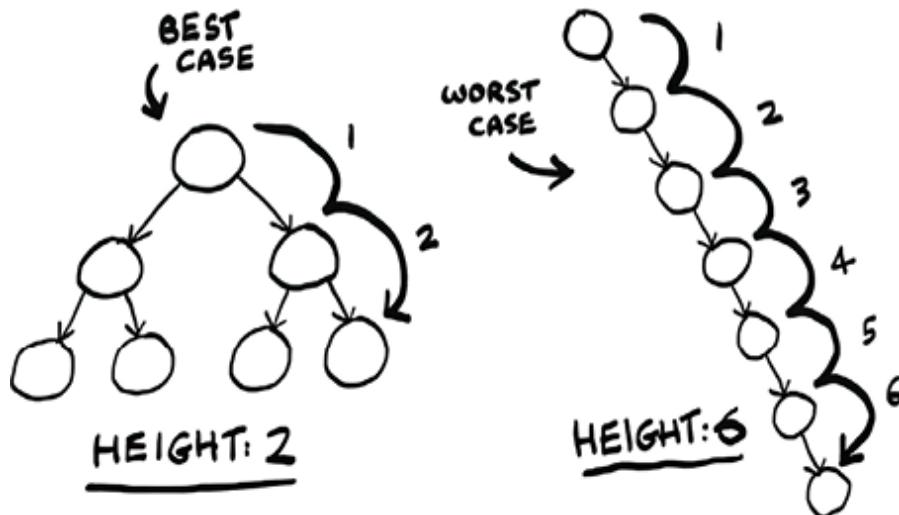
¡Excepto que ahora no está allí! Si estuviera en el árbol, estaría justo donde está el nodo punteado. La única razón por la que hablamos de árboles es para ver si son más rápidos que los arreglos y las listas enlazadas. Entonces, hablemos del rendimiento de este árbol. Para ello, debemos fijarnos en la altura del árbol.

## ***Shorter trees are faster***

## ***Los árboles más bajos son más rápidos***

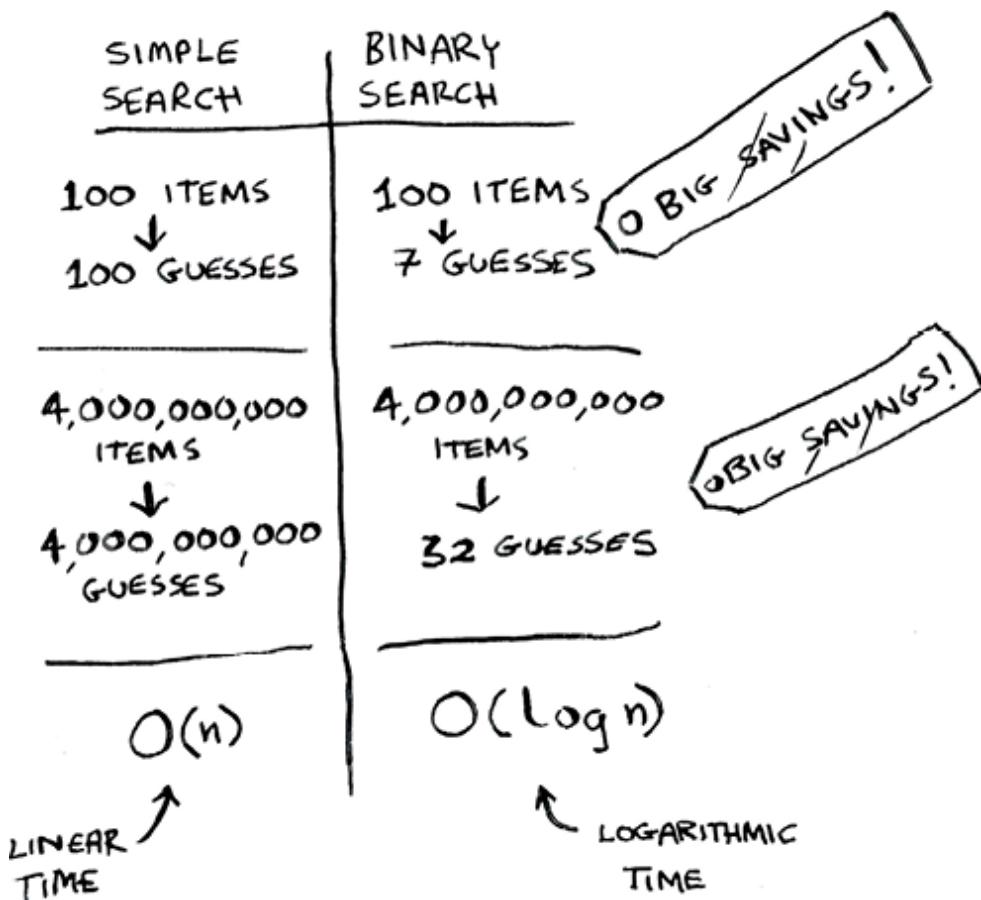
Let's look at two trees. They both have seven nodes, but the performance is very different.

Miremos dos árboles. Ambos tienen siete nodos, pero el rendimiento es muy diferente.



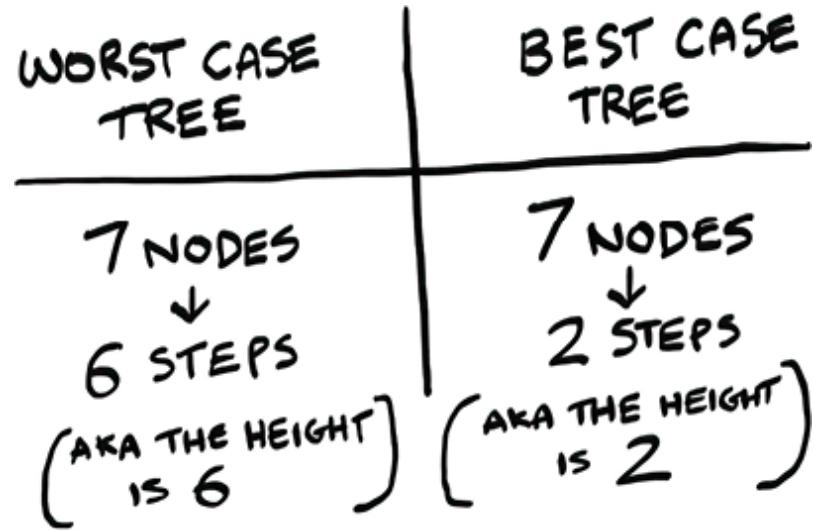
The height of the best-case tree is 2. This means you can get to any node from the root node in, at most, two steps. The height of the worst-case tree is 6. This means you can get to any node from the root node in, at most, *six* steps. Let's compare this to the performance of binary search versus simple search. Just to remind you, here's the performance of binary search versus simple search:

La altura del árbol en el mejor de los casos es 2. Esto significa que puede llegar a cualquier nodo desde el nodo raíz en, como máximo, dos pasos. La altura del árbol en el peor de los casos es 6. Esto significa que puede llegar a cualquier nodo desde el nodo raíz en, como máximo, seis pasos. Comparemos esto con el rendimiento de la búsqueda binaria versus la búsqueda simple. Sólo para recordarle, este es el rendimiento de la búsqueda binaria versus la búsqueda simple:



Remember our number guessing game? To guess a number out of 100 numbers, it would take 7 guesses with binary search, but 100 guesses with simple search. Well, we're in a similar situation with trees.

¿Recuerdas nuestro juego de adivinar números? Para adivinar un número entre 100 números, se necesitarían 7 conjeturas con la búsqueda binaria, pero 100 conjeturas con la búsqueda simple. Bueno, estamos en una situación similar con los árboles.



The worst-case tree is taller, and it has worse performance. In the worst-case tree, the nodes are all in a line. This tree has height  $O(n)$ , so searches will take  $O(n)$  time. You can think of it this way: this tree is really just a linked list since one node points to another, and so on, in a line. And searching through a linked list takes  $O(n)$  time.

El árbol en el peor de los casos es más alto y tiene peor rendimiento. En el peor de los casos, los nodos están todos en línea. Este árbol tiene una altura  $O(n)$ , por lo que las búsquedas tomarán un tiempo  $O(n)$ . Puedes verlo de esta manera: este árbol es en realidad sólo una lista enlazada ya que un nodo apunta a otro, y así sucesivamente, en una línea. Y buscar en una lista vinculada lleva  $O(n)$  tiempo.

The best-case tree has height  $O(\log n)$ , and searching this tree will take  $O(\log n)$  time.

El árbol en el mejor de los casos tiene una altura  $O(\log n)$ , y buscar en este árbol llevará un tiempo  $O(\log n)$ .

WORST CASE TREE	BEST CASE TREE
7 NODES ↓ 6 STEPS	7 NODES ↓ 2 STEPS
SEARCH TIME:	$O(n)$
	$O(\log n)$

So this situation is very similar to binary search versus simple search! *If we can guarantee the height of our tree will be  $O(\log n)$ , then searching the tree will be  $O(\log n)$ , just like we wanted.*

Entonces esta situación es muy similar a la búsqueda binaria versus la búsqueda simple! Si podemos garantizar que la altura de nuestro árbol será  $O(\log n)$ , entonces la búsqueda en el árbol será  $O(\log n)$ , tal como queríamos.

	SEARCH	INSERT
SORTED ARRAY	$O(\log n)$	$O(n)$
LINKED LIST	$O(n)$	$O(1)$
???	$O(\log n)$	FASTER THAN $O(n)$

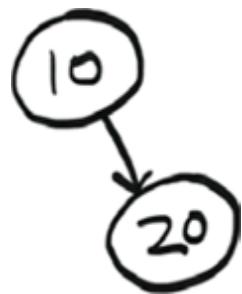
But how can we guarantee the height will be  $O(\log n)$ ? Here's an example where we build a tree that ends up being a worst-case tree (something we want to avoid). We will start with one node.

Pero, ¿cómo podemos garantizar que la altura será  $O(\log n)$ ? Aquí hay un ejemplo en el que construimos un árbol que termina siendo el peor de los casos (algo que queremos evitar). Comenzaremos con un nodo.



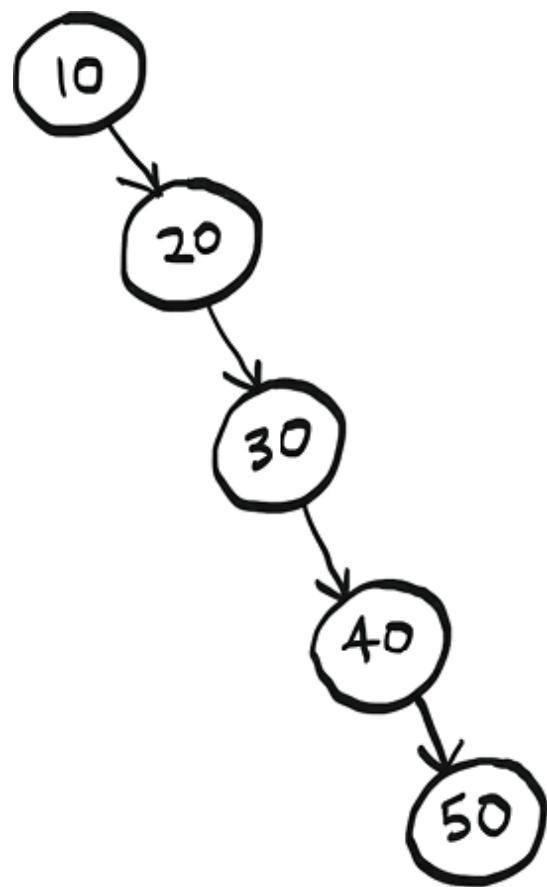
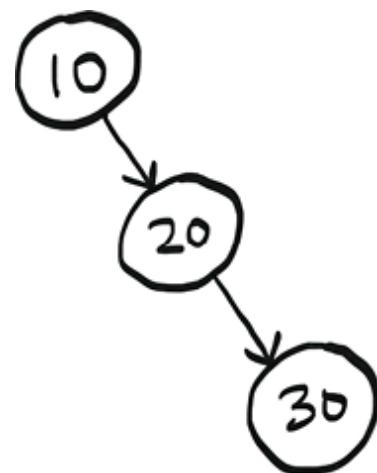
Let's add another one.

Agreguemos otro.



So far, so good. Let's add a few more.

Hasta ahora, todo bien. Agreguemos algunos más.



We keep having to add these nodes to the right since they are all bigger than the other nodes.

Seguimos teniendo que agregar estos nodos a la derecha ya que todos son más grandes que los otros nodos.

Yikes, this tree is a worst-case tree—its height is  $O(n)$ !  
Shorter trees are faster. The shortest a BST can be is  $O(\log n)$ . To make a BST shorter, we need to balance it. So lets look at a balanced BST next.

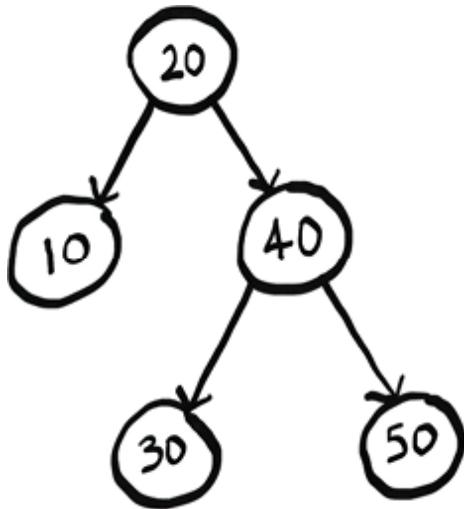
Vaya, este árbol es el peor de los casos: su altura es  $O(n)$ !  
Los árboles más bajos son más rápidos. Lo más corto que puede ser un BST es  $O(\log n)$ . Para acortar un BST, debemos equilibrarlo. Así que veamos a continuación un BST equilibrado.

## ***AVL trees: A type of balanced tree***

### ***Árboles AVL: un tipo de árbol equilibrado***

AVL trees are a type of self-balancing BST. This means AVL trees will maintain a height of  $O(\log n)$ . Whenever the tree is out of balance—that is, the height is not  $O(\log n)$ —it will correct itself. For the last example, the tree may balance itself to look like this.

Los árboles AVL son un tipo de BST autoequilibrado. Esto significa que los árboles AVL mantendrán una altura de  $O(\log n)$ . Siempre que el árbol esté desequilibrado, es decir, la altura no sea  $O(\log n)$ , se corregirá solo. En el último ejemplo, el árbol puede equilibrarse para verse así.



An AVL tree will give us that  $O(\log n)$  height we want by balancing itself through rotations.

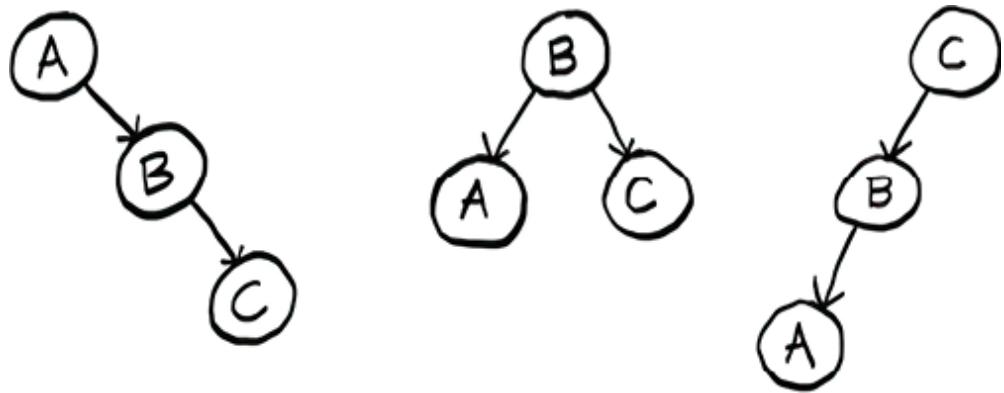
Un árbol AVL nos dará la altura  $O(\log n)$  que queremos equilibrándose mediante rotaciones.

## Rotations

### Rotaciones

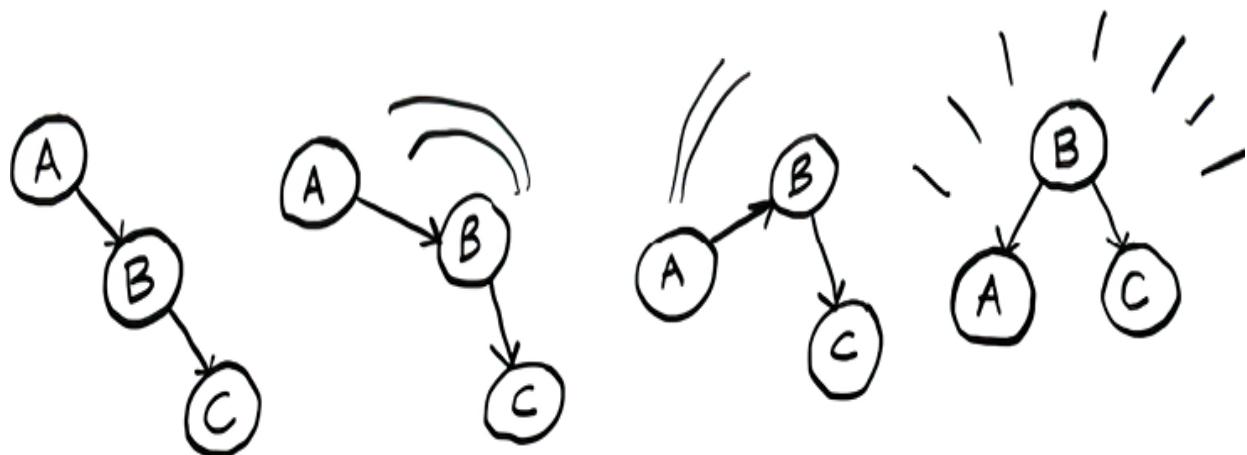
Suppose you have a tree with three nodes. Any one of them could be the root node.

Supongamos que tiene un árbol con tres nodos. Cualquiera de ellos podría ser el nodo raíz.



You use rotation to move a set of nodes to end up with a new arrangement. Let's see a rotation in slow motion.

Utiliza la rotación para mover un conjunto de nodos y terminar con una nueva disposición. Veamos una rotación en cámara lenta.



We rotated to the left. We started with an unbalanced tree with A as the root node and ended up with a balanced tree with B as the root node.

Giramos hacia la izquierda. Comenzamos con un árbol desequilibrado con A como nodo raíz y terminamos con un árbol equilibrado con B como nodo raíz.

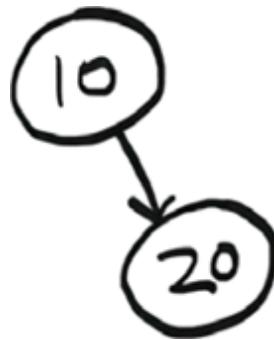
Rotations are a popular way to balance trees. AVL trees use rotation to balance. Let's see an example. We'll start with one node again.

Las rotaciones son una forma popular de equilibrar los árboles. Los árboles AVL utilizan la rotación para equilibrarse. Veamos un ejemplo. Empezaremos con un nodo nuevamente.



Let's add a node.

Agreguemos un nodo.



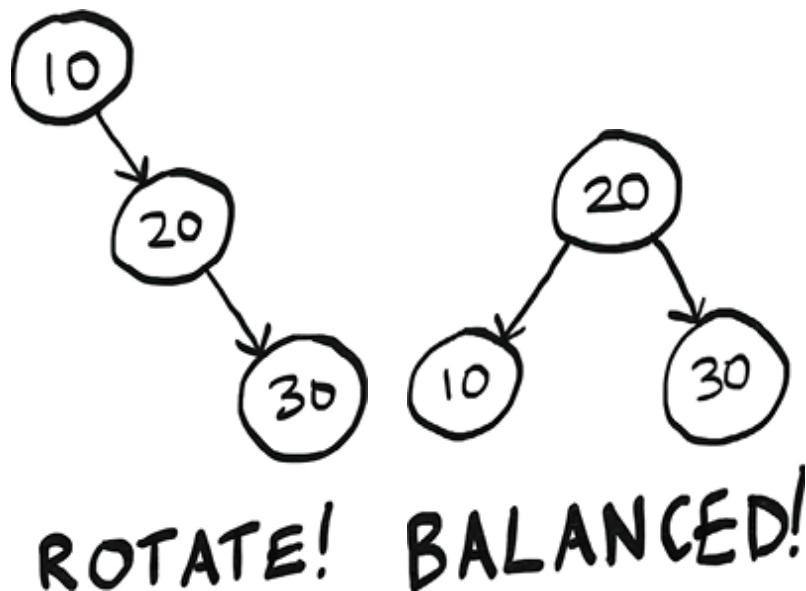
So far, so good. The children aren't exactly the same height; there's a difference of 1. But a difference of 1 is OK for AVL trees. Now we will add one more.

Hasta ahora, todo bien. Los niños no tienen exactamente la misma altura; hay una diferencia de 1. Pero una diferencia

de 1 está bien para los árboles AVL. Ahora agregaremos uno más.

Uh oh! Now the tree is unbalanced. Time to rotate!

¡UH oh! Ahora el árbol está desequilibrado. ¡Es hora de rotar!

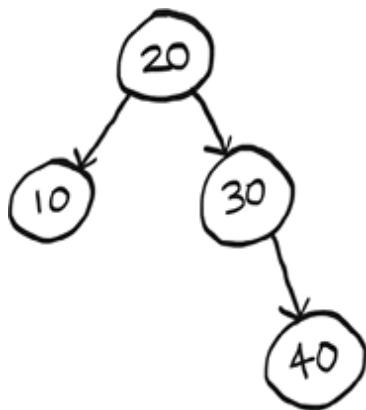


We did a left rotation, and now the tree is balanced again.

Hicimos una rotación hacia la izquierda y ahora el árbol vuelve a estar equilibrado.

Let's add one more node.

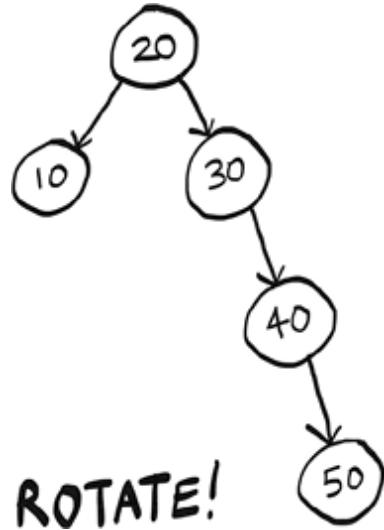
Agreguemos un nodo más.



DIFFERENCE  
OF 1 IS OK

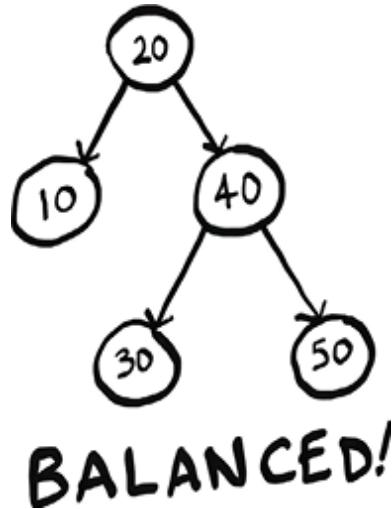
And add another node.

Y agregue otro nodo.



We need to rotate again!

Necesitamos rotar de nuevo!



Phew. Using rotation, the AVL tree has balanced itself. Notice in that last example, the node "30" got rotated instead of the node "20". There's an example coming up that explains why.

Uf. Mediante la rotación, el árbol AVL se ha equilibrado. Observe que en el último ejemplo, se rotó el nodo "30" en lugar del nodo "20". Se viene un ejemplo que explica por qué.

**How does the AVL tree know when it's time to rotate?**

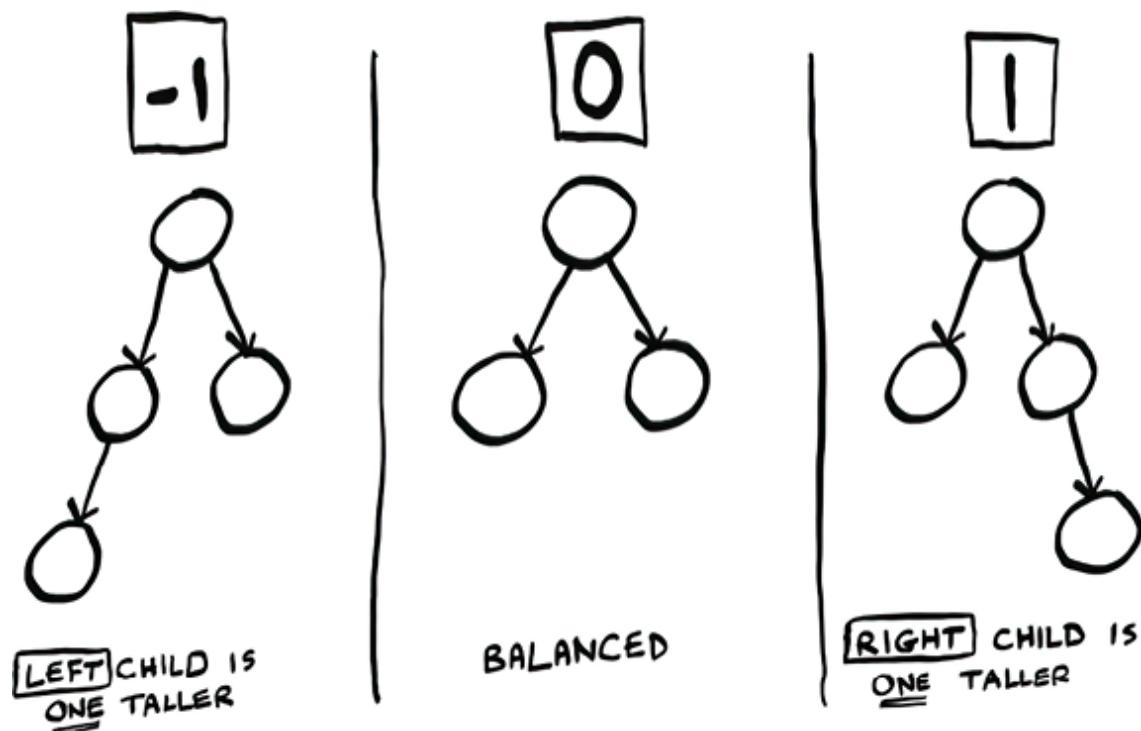
**¿Cómo sabe el árbol AVL cuándo es el momento de rotar?**

We can see visually that the tree is off balance—one side is longer than the other. But how does the tree know that?

Podemos ver visualmente que el árbol está desequilibrado: un lado es más largo que el otro. ¿Pero cómo sabe eso el árbol?

For the tree to know when it's time to balance itself, it needs to store some extra information. Each node stores one of two pieces of information: its height or something called a *balance factor*. The balance factor can be  $-1$ ,  $0$ , or  $1$ .

Para que el árbol sepa cuándo es el momento de equilibrarse, necesita almacenar información adicional. Cada nodo almacena uno de dos datos: su altura o algo llamado factor de equilibrio. El factor de equilibrio puede ser  $-1$ ,  $0$  o  $1$ .



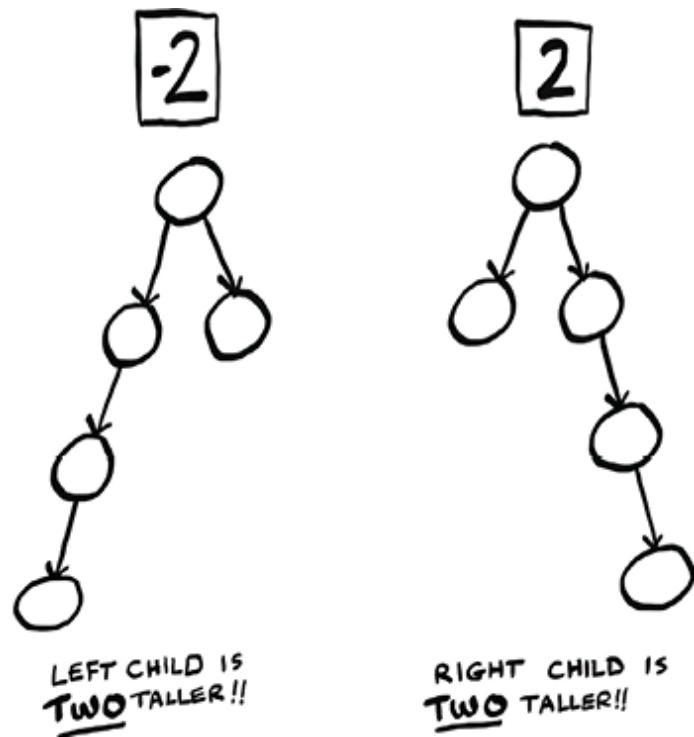
Note that this image shows you the balance factors for the root nodes only, but you would need to store the balance

factor for each node (I'll show an example of this shortly).

Tenga en cuenta que esta imagen muestra los factores de equilibrio para los nodos raíz únicamente, pero necesitará almacenar el factor de equilibrio para cada nodo (muestreará un ejemplo de esto en breve).

The balance factor tells you which child is taller and by how much. The balance factor lets the tree know when to rebalance. Zero means the tree is balanced. A  $-1$  or  $1$  is OK, too, because remember, AVL trees don't have to be perfectly balanced: a difference of one is OK.

El factor de equilibrio le indica qué niño es más alto y cuánto. El factor de equilibrio le permite al árbol saber cuándo reequilibrarse. Cero significa que el árbol está equilibrado. Un  $-1$  o  $1$  también está bien, porque recuerde, los árboles AVL no tienen que estar perfectamente equilibrados: una diferencia de uno está bien.



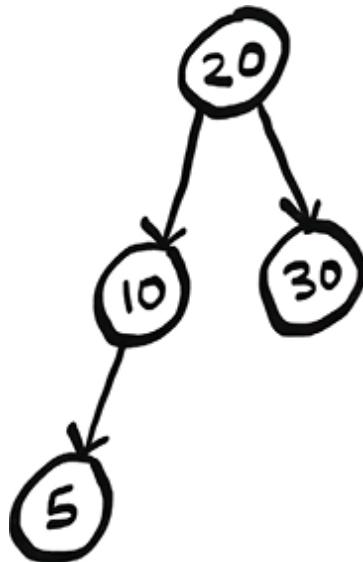
But if the balance factor drops below  $-1$  or moves above  $1$ , it's time to rebalance. To the right are two trees that need rebalancing.

Pero si el factor de equilibrio cae por debajo de  $-1$  o sube por encima de  $1$ , es hora de reequilibrar. A la derecha hay dos árboles que necesitan reequilibrarse.

As I said, each node needs to store either the height or the balance factor. In my example, I am going to store both so you can see how they both change. But if you have the heights of each subtree, it is easy to compute the balance factor. Let's see an example. Take this tree.

Como dije, cada nodo necesita almacenar la altura o el factor de equilibrio. En mi ejemplo, voy a almacenar ambos para que puedas ver cómo cambian. Pero si tienes las

alturas de cada subárbol, es fácil calcular el factor de equilibrio. Veamos un ejemplo. Toma este árbol.



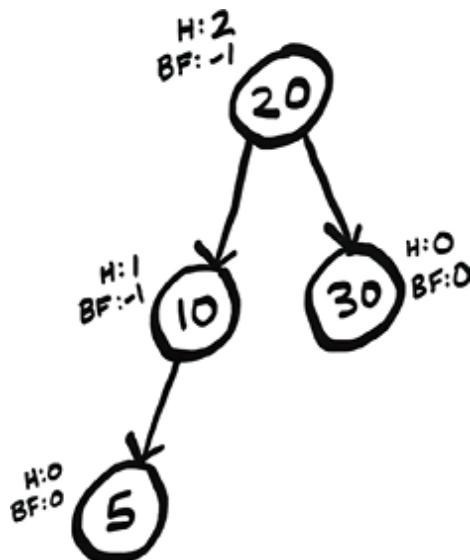
We are going to add this node to it.

Le vamos a agregar este nodo.



First, let's write out the height and balance factors for each node. In this image, H is height, and BF is balance factor.

Primero, escribamos los factores de altura y equilibrio de cada nodo. En esta imagen, H es la altura y BF es el factor de equilibrio.

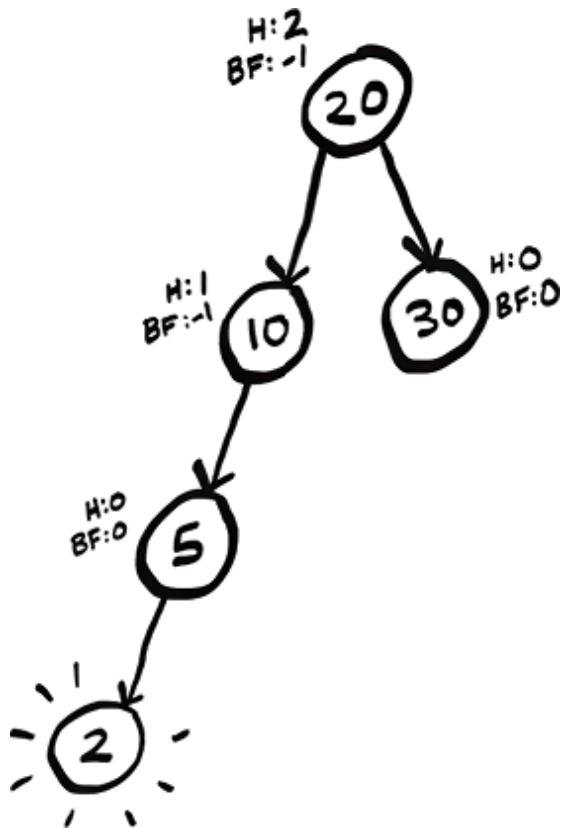


Remember, I am storing both values to show how they change, but you would only need to store one. Make sure these numbers make sense to you. Note that all the leaf nodes have a balance factor of 0: they have no child nodes, so there's nothing to keep in balance.

Recuerde, estoy almacenando ambos valores para mostrar cómo cambian, pero usted sólo necesitaría almacenar uno. Asegúrese de que estos números tengan sentido para usted. Tenga en cuenta que todos los nodos hoja tienen un factor de equilibrio de 0: no tienen nodos secundarios, por lo que no hay nada que mantener en equilibrio.

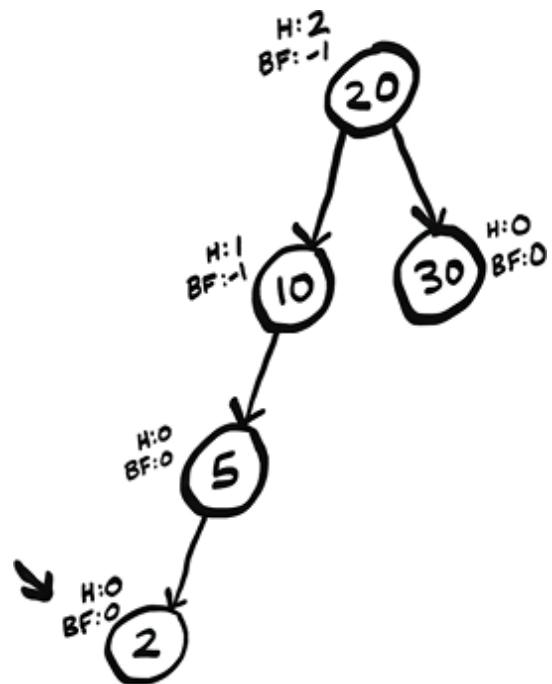
Now let's add the new node.

Ahora agreguemos el nuevo nodo.



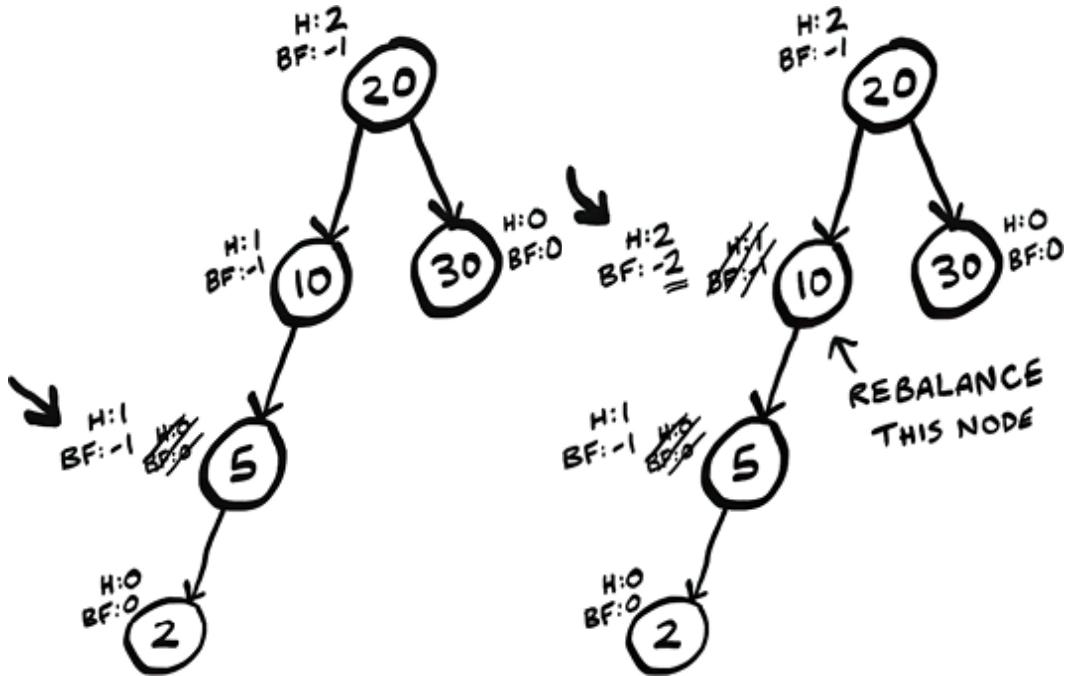
After adding this node, we need to set its height and balance factor. Then we can go up the tree, updating the heights and balance factors for all its ancestors.

Después de agregar este nodo, debemos establecer su altura y factor de equilibrio. Luego podremos subir al árbol, actualizando las alturas y factores de equilibrio de todos sus ancestros.



**Set the height and balance factor.**

**Establezca el factor de altura y equilibrio.**



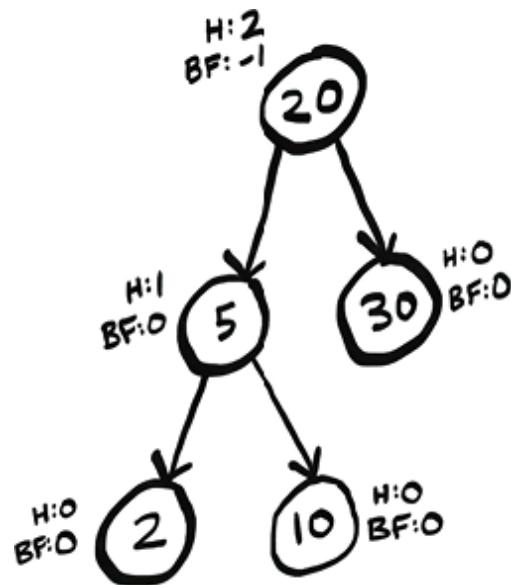
**Go up the tree and update the height and balance factor for the ancestors.**

**Sube al árbol y actualiza el factor de altura y equilibrio de los antepasados.**

Aha! We just set the balance factor to  $-2$ , which means it's time to rotate! I will show the rest of the example next, but this is the main takeaway: after an insert, you update the balance factors for that node's ancestors. The AVL tree looks at the balance factor to know when it needs to rebalance. Finishing the example, let's rotate the 10.

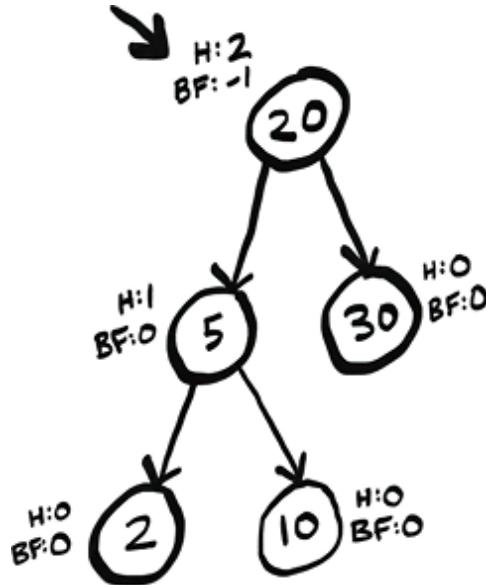
¡Ajá! Simplemente establecimos el factor de equilibrio en  $-2$ , lo que significa que es hora de rotar. Mostraré el resto del ejemplo a continuación, pero esta es la conclusión principal: después de una inserción, se actualizan los factores de equilibrio para los ancestros de ese nodo. El

árbol AVL analiza el factor de equilibrio para saber cuándo es necesario reequilibrarlo. Terminando el ejemplo, rotemos el 10.



Now that subtree is balanced. Let's keep moving up the tree.

Ahora ese subárbol está equilibrado. Sigamos subiendo por el árbol.



Nothing to update there. We actually didn't need to keep moving up the tree because AVL trees require, at most, one rebalancing.

Nada que actualizar allí. En realidad, no necesitábamos seguir ascendiendo en el árbol porque los árboles AVL requieren, como máximo, un reequilibrio.

AVL trees are a good option if you want a balanced BST. Let's recap our journey:

Los árboles AVL son una buena opción si quieres un BST equilibrado. Recapitulemos nuestro viaje:

- Binary trees are a type of tree.  
Los árboles binarios son un tipo de árbol.
- In binary trees, each node has, at most, two children.  
En los árboles binarios, cada nodo tiene, como máximo, dos hijos.

- BSTs are a type of binary tree where all the values in the left subtree are smaller than the node, and all the values in the right are greater than the node.

Los BST son un tipo de árbol binario donde todos los valores del subárbol izquierdo son más pequeños que el nodo y todos los valores de la derecha son mayores que el nodo.

- BSTs can give great performance if we can guarantee their height will be  $O(\log n)$ .

Los BST pueden ofrecer un gran rendimiento si podemos garantizar que su altura será  $O(\log n)$ .

- AVL trees are self-balancing BSTs, guaranteeing their height will be  $O(\log n)$ .

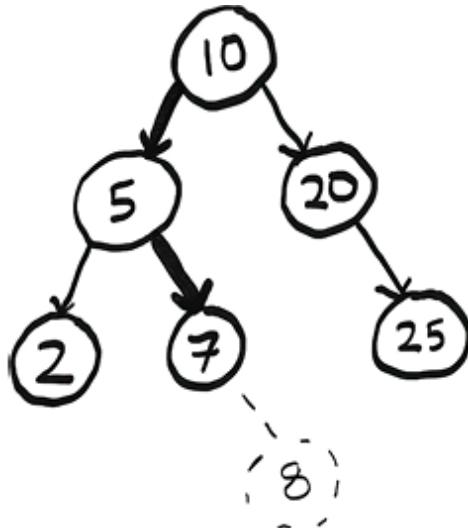
Los árboles AVL son BST autoequilibrados, lo que garantiza que su altura será  $O(\log n)$ .

- AVL trees balance themselves through rotations.

Los árboles AVL se equilibran mediante rotaciones.

We haven't covered everything. We have covered one case for rotations, and there are other cases. We won't spend time digging into them, as you will rarely need to implement an AVL tree yourself.

No hemos cubierto todo. Hemos cubierto un caso de rotaciones y hay otros casos. No dedicaremos tiempo a profundizar en ellos, ya que rara vez necesitará implementar un árbol AVL usted mismo.



Now we know AVL trees offer  $O(\log n)$  search performance. What about insertions? Well, insertions are just a matter of searching for a place to insert the node and adding a pointer, just like a linked list. For example, if we want to insert an 8 in this tree, we just need to find where to add the pointer.

Ahora sabemos que los árboles AVL ofrecen un rendimiento de búsqueda  $O(\log n)$ . ¿Qué pasa con las inserciones? Bueno, las inserciones son solo cuestión de buscar un lugar para insertar el nodo y agregar un puntero, como una lista enlazada. Por ejemplo, si queremos insertar un 8 en este árbol, sólo necesitamos encontrar dónde agregar el puntero.

So insertions are  $O(\log n)$  as well.

Entonces las inserciones también son  $O(\log n)$ .

At the start of this chapter, we were looking for a data structure that offered both fast searches and fast inserts. We have found our magic data structure: it's a balanced BST!

Al comienzo de este capítulo, buscábamos una estructura de datos que ofreciera búsquedas e inserciones rápidas. Hemos encontrado nuestra estructura de datos mágica: ¡es un BST equilibrado!

	SEARCH	INSERT
SORTED ARRAY	$O(\log n)$	$O(n)$
LINKED LIST	$O(n)$	$O(1)$
BST	$O(\log n)$	$O(\log n)$

## ***Splay trees***

### **árboles extendidos**

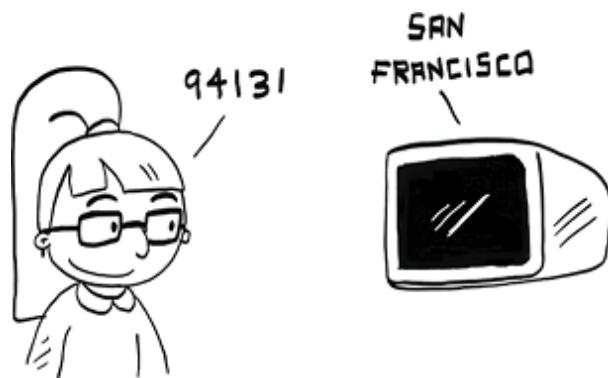
AVL trees are a good basic balanced BST that guarantees  $O(\log n)$  time for a bunch of operations.

Los árboles AVL son un buen BST básico equilibrado que garantiza un tiempo  $O(\log n)$  para un conjunto de operaciones.

Splay trees are a different take on balanced BSTs. The cool thing about splay trees is if you have recently looked up an item, the next time you look it up, the lookup will be faster. There is something intuitively pleasing about this. For

example, suppose you have some software where you give it a zip code, and it will look up the city for you.

Los árboles de expansión son una versión diferente de los BST equilibrados. Lo bueno de los árboles de distribución es que si recientemente buscó un elemento, la próxima vez que lo busque, la búsqueda será más rápida. Hay algo intuitivamente agradable en esto. Por ejemplo, supongamos que tiene algún software al que le proporciona un código postal y buscará la ciudad por usted.



You can picture the interaction going like this.

Puedes imaginarte la interacción así.



Now suppose you are repeatedly looking up the same zip code.

Ahora suponga que busca repetidamente el mismo código postal.

10:20 am

94131



LET ME  
LOOK IT  
UP...



HMM...



SAN  
FRANCISCO!

THANKS!



10:30 am

94131



LET ME  
LOOK IT  
UP...



HMM...



SAN  
FRANCISCO!

THANKS!



10:45 am

94131



LET ME  
LOOK IT  
UP...



UH... DIDN'T YOU  
JUST LOOK THIS  
UP?



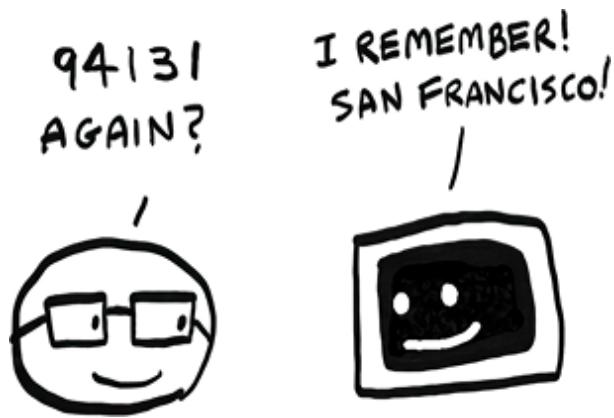
HMM...

That feels sort of silly.

Eso parece un poco tonto.

The software just looked up the zip code; why can't it remember that? It should really go something like this.

El software simplemente buscó el código postal; ¿Por qué no puede recordar eso? Realmente debería ser algo como esto.



This is what splay trees allow you to do. When you look up a node in a splay tree, it will make that node the new root, so if you look it up again, the lookup will be instant. In general, the nodes you have looked up recently get clustered to the top and become faster to look up.

Esto es lo que los árboles extendidos te permiten hacer. Cuando busca un nodo en un árbol extendido, ese nodo se convertirá en la nueva raíz, por lo que si lo busca nuevamente, la búsqueda será instantánea. En general, los nodos que ha buscado recientemente se agrupan en la parte superior y su búsqueda es más rápida.

The tradeoff is the tree is not guaranteed to be balanced. So *some* searches might take longer than  $O(\log n)$  time. Some searches may take as long as linear time! Also, while performing the search, you may have to rotate the node up to the root if it is not already the root, and that will take time.

La desventaja es que no se garantiza que el árbol esté equilibrado. Por lo tanto, algunas búsquedas pueden tardar más que  $O(\log n)$ . ¡Algunas búsquedas pueden tardar tanto como el tiempo lineal! Además, mientras realiza la búsqueda, es posible que tenga que rotar el nodo hasta la raíz si aún no es la raíz, y eso llevará tiempo.

But we are OK with the tradeoff of not having a balanced tree all the time. Because the cool thing is that if you do  $n$  searches, the total time is  $O(n \log n)$  guaranteed—that is,  $O(\log n)$  per search. So even though a single search may take longer than  $O(\log n)$  time, overall, they will average out to  $O(\log n)$  time, and the faster search time is our goal.

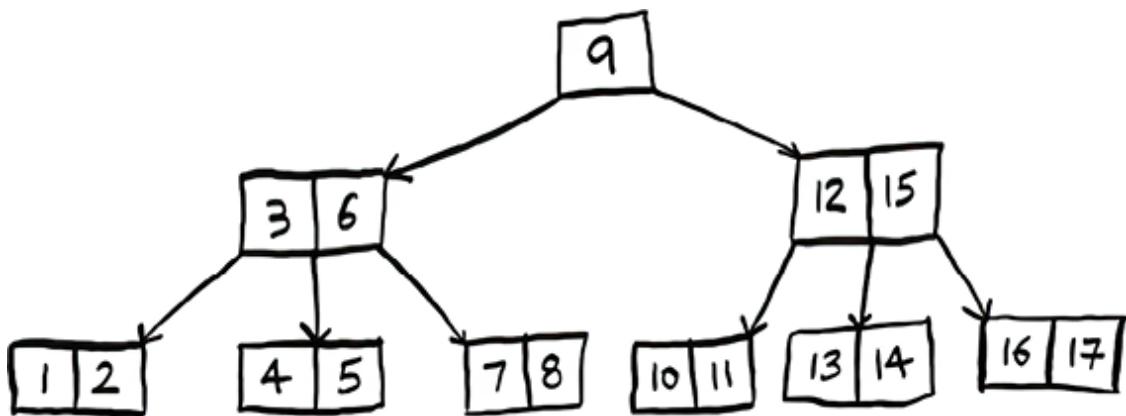
Pero estamos de acuerdo con la desventaja de no tener un árbol equilibrado todo el tiempo. Porque lo bueno es que si haces  $n$  búsquedas, el tiempo total está garantizado  $O(n \log n)$ , es decir,  $O(\log n)$  por búsqueda. Entonces, aunque una sola búsqueda puede tomar más tiempo que  $O(\log n)$ , en general, promediarán el tiempo  $O(\log n)$ , y nuestro objetivo es un tiempo de búsqueda más rápido.

## **B-trees**

### **árboles B**

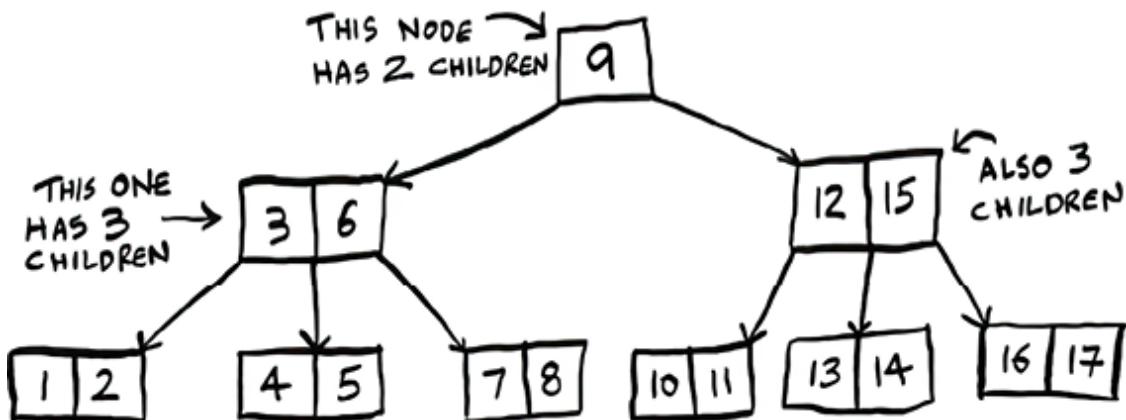
B-trees are a generalized form of binary tree. They are often used for building databases. Here is a B-tree.

Los árboles B son una forma generalizada de árbol binario. A menudo se utilizan para crear bases de datos. Aquí hay un árbol B.



Looks pretty wild, huh? You may notice that some of these nodes have more than two children.

Parece bastante salvaje, ¿eh? Puede notar que algunos de estos nodos tienen más de dos hijos.

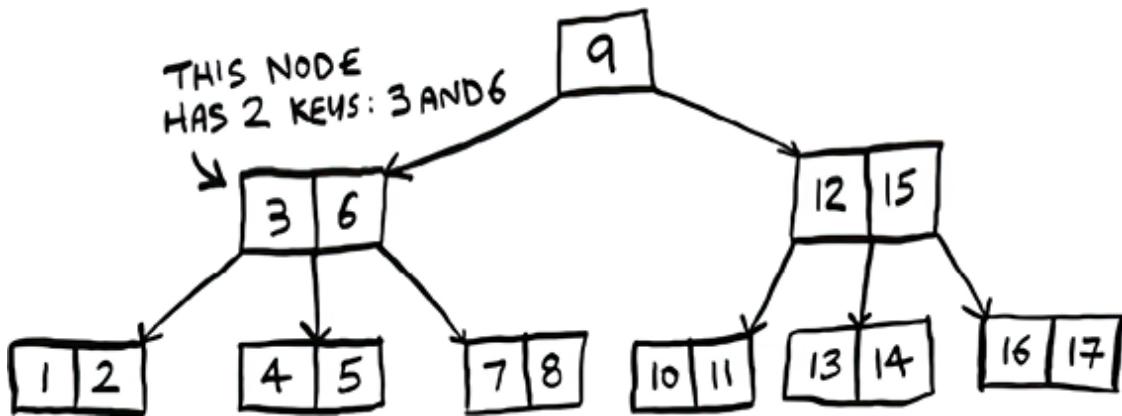


Unlike binary trees, B-trees can have many more children.

A diferencia de los árboles binarios, los árboles B pueden tener muchos más hijos.

You've probably also noticed that, unlike our previous trees, most nodes have two keys.

Probablemente también hayas notado que, a diferencia de nuestros árboles anteriores, la mayoría de los nodos tienen dos claves.



So not only can nodes in B-trees have more than two children, they can have more than one key! This is why I said B-trees are a generalized form of BSTs.

Entonces, los nodos en los árboles B no solo pueden tener más de dos hijos, sino que también pueden tener más de una clave! Por eso dije que los árboles B son una forma generalizada de BST.

## What is the advantage of B-trees?

## ¿Cuál es la ventaja de los árboles B?

B-trees have a very interesting optimization because it's a physical optimization. Computers are physical objects. So

when we are looking things up in a tree, a physical object has to move to retrieve that data. This is called *seek time*. Seek time can be a big factor in how fast or slow your algorithm is.

Los árboles B tienen una optimización muy interesante porque es una optimización física. Las computadoras son objetos físicos. Entonces, cuando buscamos cosas en un árbol, un objeto físico tiene que moverse para recuperar esos datos. Esto se llama tiempo de búsqueda. El tiempo de búsqueda puede ser un factor importante en la rapidez o lentitud de su algoritmo.

Think of it like going to the grocery store. You could go shopping for one item at a time. Suppose you decide to buy milk. After coming home, you realize you should get some bread, too, so you go back to the store. After coming home again, you realize you're out of coffee. So you go back to the store again. What an inefficient way to shop! It would be much better to shop once and buy a bunch of stuff while you are there. In this example, driving to and from the store is the seek time.

Piense en ello como si fuera al supermercado. Podrías ir a comprar un artículo a la vez. Supongamos que decides comprar leche. Después de llegar a casa, te das cuenta de que también deberías comprar algo de pan, así que vuelves a la tienda. Después de volver a casa, te das cuenta de que te has quedado sin café. Así que vuelves a la tienda otra vez. ¡Qué manera tan ineficiente de comprar! Sería mucho mejor comprar una vez y comprar un montón de cosas

mientras estás allí. En este ejemplo, conducir hacia y desde la tienda es el momento de búsqueda.

The fundamental idea with B-Trees is that *once you've done the seek, you might as well read a bunch of stuff into memory*. That is, once you're at the store, you might as well buy everything you need instead of going back repeatedly.

La idea fundamental con B-Trees es que una vez que hayas realizado la búsqueda, también puedes leer un montón de cosas en la memoria. Es decir, una vez que estés en la tienda, también puedes comprar todo lo que necesitas en lugar de volver repetidamente.

B-trees have bigger nodes: each node can have many more keys and children than a binary tree. So we spend more time reading each node. *But we seek less because we read more data in one go*. This is what makes B-trees faster.

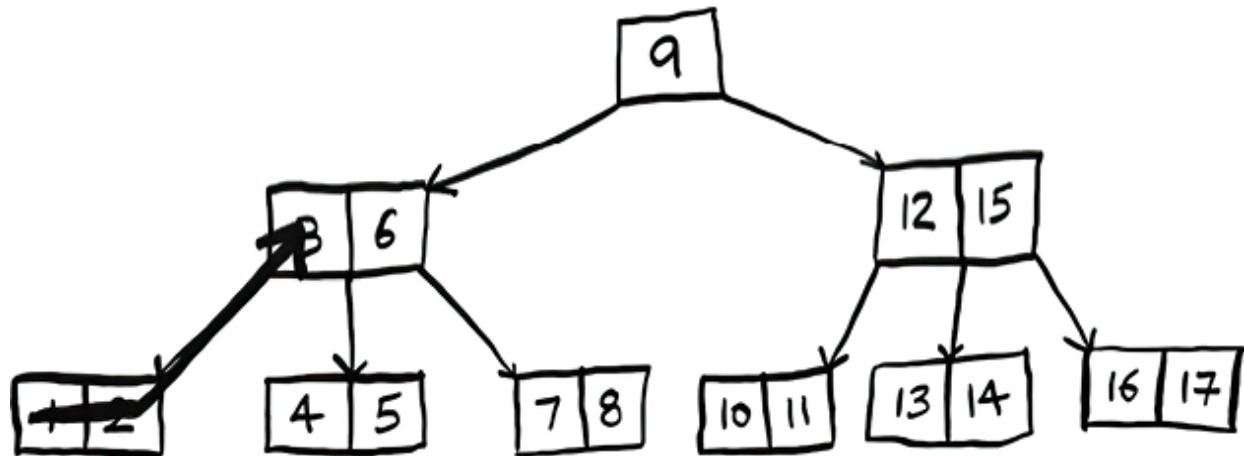
Los árboles B tienen nodos más grandes: cada nodo puede tener muchas más claves e hijos que un árbol binario. Entonces dedicamos más tiempo a leer cada nodo. Pero buscamos menos porque leemos más datos de una vez. Esto es lo que hace que los árboles B sean más rápidos.

B-trees are a popular data structure for databases, which is no surprise as databases spend a lot of time retrieving data from disk.

Los árboles B son una estructura de datos popular para bases de datos, lo cual no sorprende ya que las bases de datos dedican mucho tiempo a recuperar datos del disco.

Notice the ordering in a B-tree; it is pretty interesting, too.  
You start at the lower left.

Observe el orden en un árbol B; También es bastante interesante. Empiezas en la parte inferior izquierda.

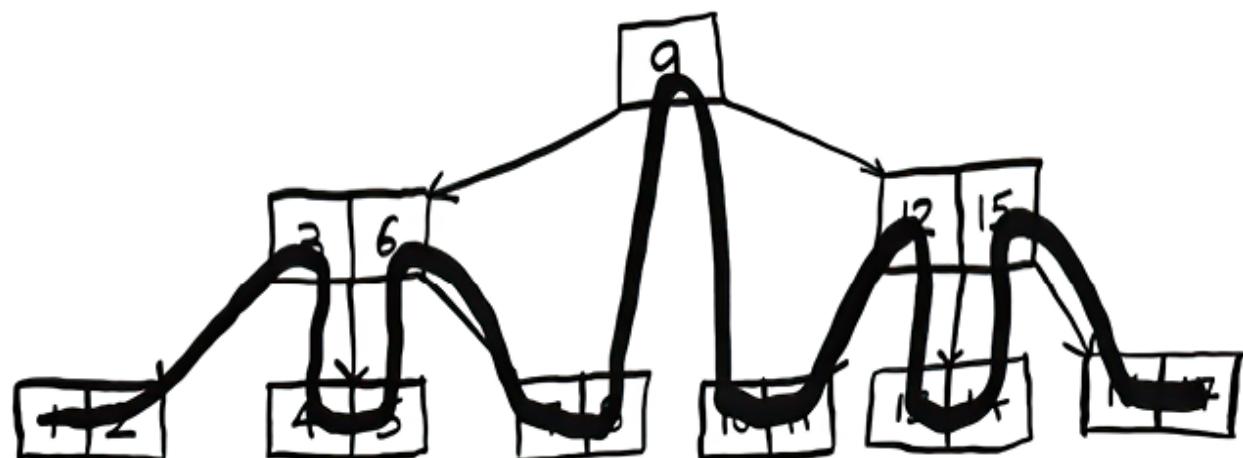
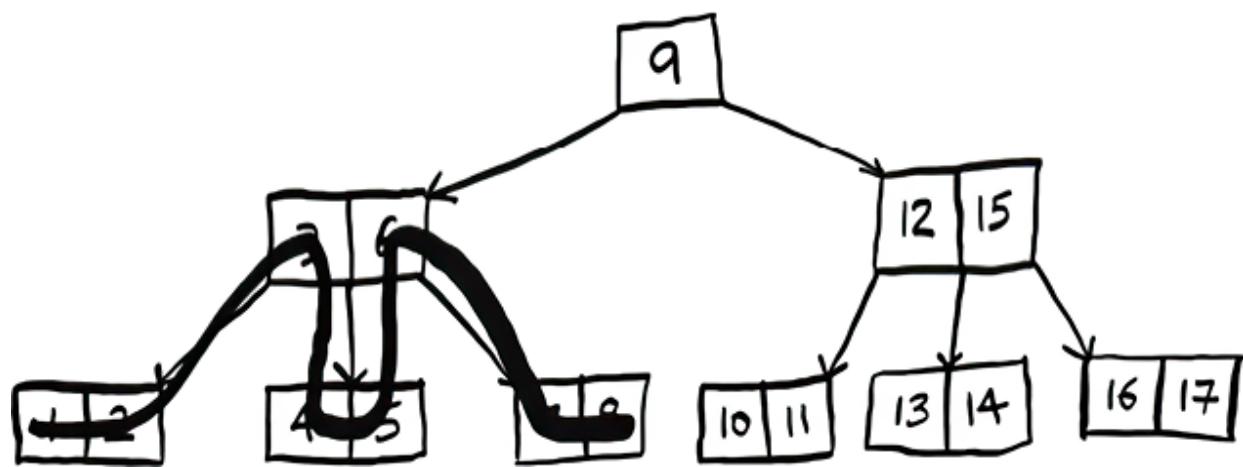
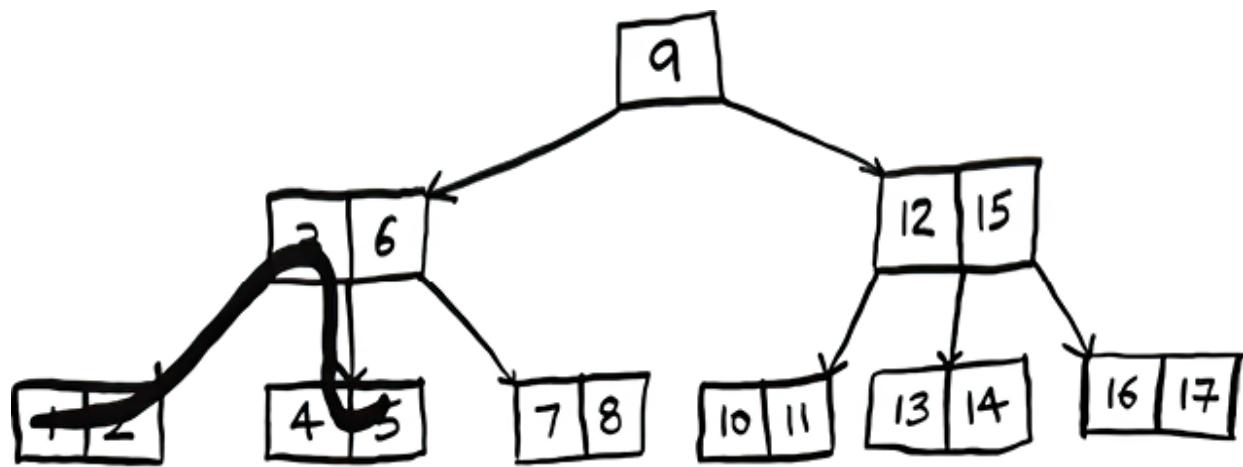


Where do you go from here?

¿A donde vas desde aquí?

You'll snake across the whole tree.

Serpearás por todo el árbol.



Notice that it is still following the property of the BST, where for each key, the keys to the left are smaller, and the keys to

the right are bigger. For example, for key 3, the keys to the left are 1 and 2, and the keys to the right are 4 and 5.

Observe que todavía sigue la propiedad de BST, donde para cada clave, las claves de la izquierda son más pequeñas y las claves de la derecha son más grandes. Por ejemplo, para la tecla 3, las teclas de la izquierda son 1 y 2, y las teclas de la derecha son 4 y 5.

Also notice that the number of children is one greater than the number of keys. So the root node has one key and two children. Each of those children has two keys and three children.

Observe también que el número de hijos es uno mayor que el número de claves. Entonces el nodo raíz tiene una clave y dos hijos. Cada uno de esos niños tiene dos llaves y tres hijos.

That concludes our two chapters on trees. It's unlikely you will need to implement a tree yourself, but it's important to know that trees are a type of graph, and they offer great performance. In the next chapter, we will return to graphs and talk about a new type of graph: a weighted graph.

Con esto concluyen nuestros dos capítulos sobre árboles. Es poco probable que necesite implementar un árbol usted mismo, pero es importante saber que los árboles son un tipo de gráfico y ofrecen un gran rendimiento. En el próximo capítulo, volveremos a los gráficos y hablaremos sobre un nuevo tipo de gráfico: un gráfico ponderado.

## **Recap**

### **Resumen**

- Balanced binary search trees (BSTs) offer the same Big O search performance as arrays with better insertion performance.

Los árboles de búsqueda binaria equilibrados (BST) ofrecen el mismo rendimiento de búsqueda Big O que las matrices con mejor rendimiento de inserción.
- The height of a tree affects its performance.

La altura de un árbol afecta su desempeño.
- AVL trees are a popular type of balanced BST. Like most balanced trees, AVL trees balance themselves through rotation.

Los árboles AVL son un tipo popular de BST equilibrado. Como la mayoría de los árboles equilibrados, los árboles AVL se equilibran mediante la rotación.
- B-trees are generalized BSTs, where each node can have multiple keys and multiple child nodes.

Los árboles B son BST generalizados, donde cada nodo puede tener múltiples claves y múltiples nodos secundarios.

- Seek time is like travel time to a grocery store. B-trees try to minimize seek time by reading more data in one go.

Buscar tiempo es como el tiempo de viaje a una tienda de comestibles. Los árboles B intentan minimizar el tiempo de búsqueda leyendo más datos de una sola vez.



# **9 Dijkstra's algorithm**

---

## **9 algoritmo de Dijkstra**

---

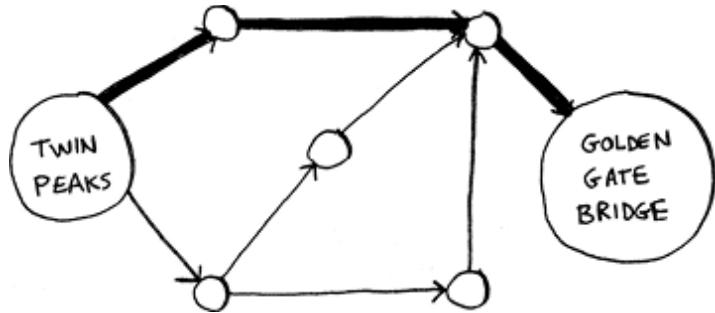
### **In this chapter**

#### **En este capítulo**

- We continue the discussion of graphs, and you learn about weighted graphs, a way to assign more or less weight to some edges.  
Continuamos la discusión sobre gráficos y aprenderá sobre gráficos ponderados, una forma de asignar más o menos peso a algunos bordes.
- You learn Dijkstra's algorithm, which lets you answer "What's the shortest path to X?" for weighted graphs.  
Aprendes el algoritmo de Dijkstra, que te permite responder "¿Cuál es el camino más corto hacia X?" para gráficos ponderados.
- You learn about negative-weight edges in graphs, where Dijkstra's algorithm doesn't work.  
Aprende sobre los bordes de peso negativo en los gráficos, donde el algoritmo de Dijkstra no funciona.

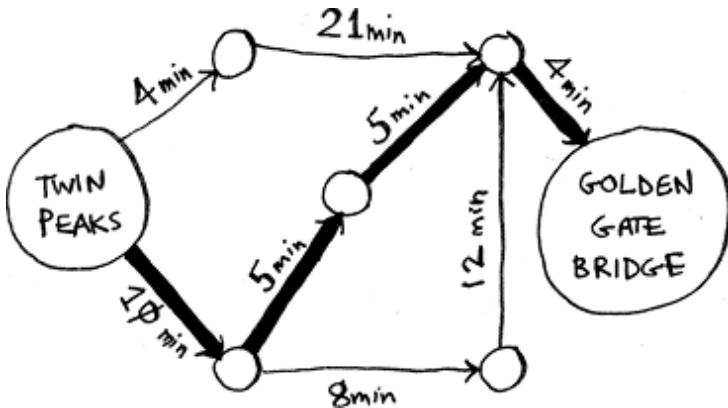
In chapter 6, you figured out a way to get from point A to point B.

En el capítulo 6, descubriste una manera de llegar del punto A al punto B.



It's not necessarily the fastest path. It's the shortest path because it has the least number of segments (three segments). But suppose you add travel times to those segments. Now you see that there's a faster path.

No es necesariamente el camino más rápido. Es el camino más corto porque tiene el menor número de segmentos (tres segmentos). Pero supongamos que agrega tiempos de viaje a esos segmentos. Ahora ves que hay un camino más rápido.



You used breadth-first search in chapter 6. Breadth-first search will find the path with the fewest segments (the first graph shown here). What if you want the fastest path instead (the second graph)? You can do that *fastest* with a different algorithm called *Dijkstra's algorithm*.

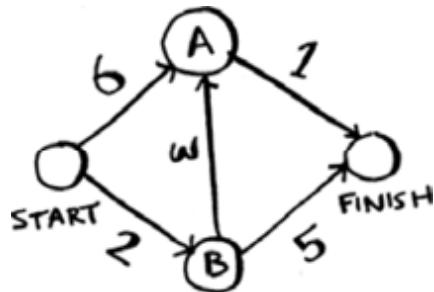
Usó la búsqueda en amplitud en el capítulo 6. La búsqueda en amplitud encontrará la ruta con la menor cantidad de segmentos (el primer gráfico que se muestra aquí). ¿Qué pasa si quieres la ruta más rápida (el segundo gráfico)? Puedes hacerlo más rápido con un algoritmo diferente llamado algoritmo de Dijkstra.

## ***Working with Dijkstra's algorithm***

### ***Trabajando con el algoritmo de Dijkstra***

Let's see how it works with this graph.

Veamos cómo funciona con este gráfico.

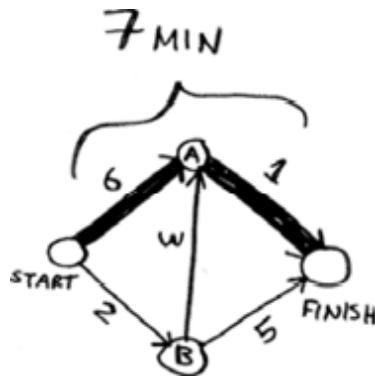


Each segment has a travel time in minutes. You'll use Dijkstra's algorithm to go from Start to Finish in the shortest possible time.

Cada segmento tiene un tiempo de viaje en minutos. Utilizarás el algoritmo de Dijkstra para ir del principio al fin en el menor tiempo posible.

If you ran breadth-first search on this graph, you'd get this shortest path.

Si realizara una búsqueda amplia en este gráfico, obtendría el camino más corto.



But that path takes 7 minutes. Let's see if you can find a path that takes less time! There are four steps to Dijkstra's algorithm:

Pero ese camino toma 7 minutos. ¡Veamos si puedes encontrar un camino que lleve menos tiempo! El algoritmo de Dijkstra consta de cuatro pasos:

1. Find the “cheapest” node. This is the node you can get to in the least amount of time.

Encuentre el nodo "más barato". Este es el nodo al que puedes llegar en la menor cantidad de tiempo.

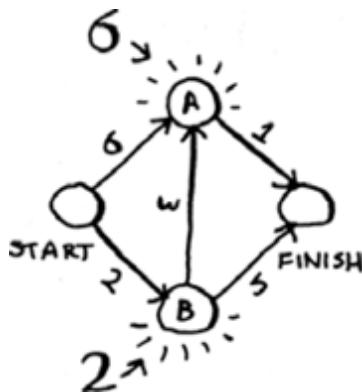
2. Update the costs of the out-neighbors of this node. I'll explain what I mean by this shortly.

Actualizar los costos de los vecinos externos de este nodo. Explicaré lo que quiero decir con esto en breve.

3. Repeat until you've done this for every node in the graph.  
Repita hasta que haya hecho esto para cada nodo del gráfico.
4. Calculate the final path.  
Calcula el camino final.

**Step 1:** Find the cheapest node. You're standing at Start, wondering if you should go to node A or node B. How long does it take to get to each node?

Paso 1: Encuentra el nodo más barato. Estás parado en Inicio y te preguntas si debes ir al nodo A o al nodo B. ¿Cuánto tiempo lleva llegar a cada nodo?



It takes 6 minutes to get to node A and 2 minutes to get to node B. The rest of the nodes, you don't know yet.

Se necesitan 6 minutos para llegar al nodo A y 2 minutos para llegar al nodo B. El resto de los nodos aún no los conoces.

Because you don't know how long it takes to get to Finish yet, you put down infinity (you'll see why soon). Node B is the closest node—it's 2 minutes away.

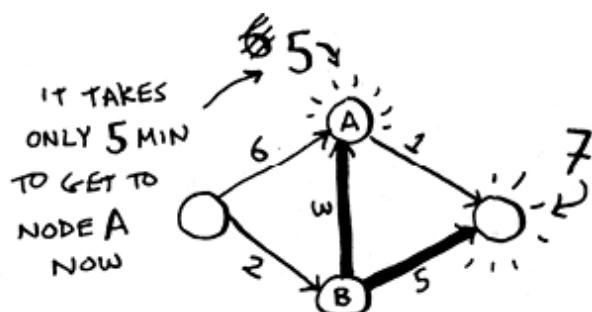
Como todavía no sabes cuánto tiempo lleva llegar a Finalizar, escribes infinito (pronto verás por qué). El nodo B es el nodo más cercano: está a 2 minutos.

NODE	TIME TO NODE
A	6
B	2
FINISH	$\infty$

**Step 2:** Calculate how long it takes to get to all of node B's out-neighbors *by following an edge from B*.

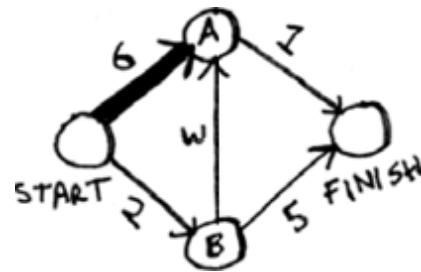
Paso 2: Calcule cuánto tiempo lleva llegar a todos los vecinos externos del nodo B siguiendo un borde desde B.

NODE	TIME
A	6
B	2
FINISH	7



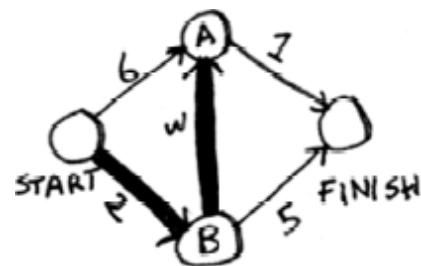
Hey, you just found a shorter path to node A! It used to take 6 minutes to get to node A.

iOye, acabas de encontrar una ruta más corta al nodo A!  
Solía tardar 6 minutos en llegar al nodo A.



But if you go through node B, there's a path that only takes 5 minutes!

Pero si pasas por el nodo B, ¡hay un camino que sólo te llevará 5 minutos!



When you find a shorter path for a neighbor of B, update its cost. In this case, you found

Cuando encuentre un camino más corto para un vecino de B, actualice su costo. En este caso, encontraste

- A shorter path to A (down from 6 minutes to 5 minutes)  
Un camino más corto hasta A (de 6 minutos a 5 minutos)

- A shorter path to Finish (down from infinity to 7 minutes)

Un camino más corto para finalizar (de infinito a 7 minutos)

**Step 3:** Repeat.

Paso 3: repetir.

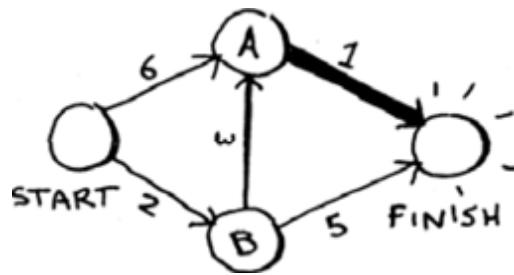
**Step 1 again:** Find the node that takes the least amount of time to get to. You're done with node B, so node A has the next smallest time estimate.

Paso 1 nuevamente: busque el nodo al que le lleve menos tiempo llegar. Ya terminaste con el nodo B, por lo que el nodo A tiene la siguiente estimación de tiempo más pequeña.

NODE	TIME
A	5
B	2
FINISH	7

**Step 2 again:** Update the costs for node A's out-neighbors.

Paso 2 nuevamente: actualice los costos de los vecinos externos del nodo A.



Woo, it takes 6 minutes to get to Finish now!

iGuau, se necesitan 6 minutos para llegar a Finalizar ahora!

You've run Dijkstra's algorithm for every node (you don't need to run it for the Finish node). At this point, you know

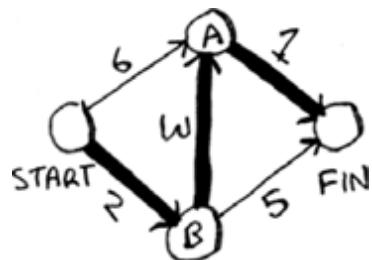
Ha ejecutado el algoritmo de Dijkstra para cada nodo (no es necesario ejecutarlo para el nodo Finalizar). En este punto, ya sabes

- It takes 2 minutes to get to node B.  
Se necesitan 2 minutos para llegar al nodo B.
- It takes 5 minutes to get to node A.  
Se necesitan 5 minutos para llegar al nodo A.
- It takes 6 minutes to get to Finish.  
Se necesitan 6 minutos para llegar a Finalizar.

NODE	TIME
A	5
B	2
FINISH	6

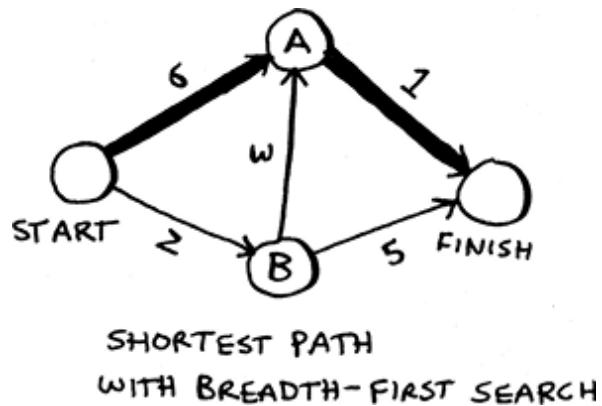
I'll save the last step, calculating the final path, for the next section. For now, I'll just show you what the final path is.

Guardaré el último paso, calcular la ruta final, para la siguiente sección. Por ahora, sólo les mostraré cuál es el camino final.



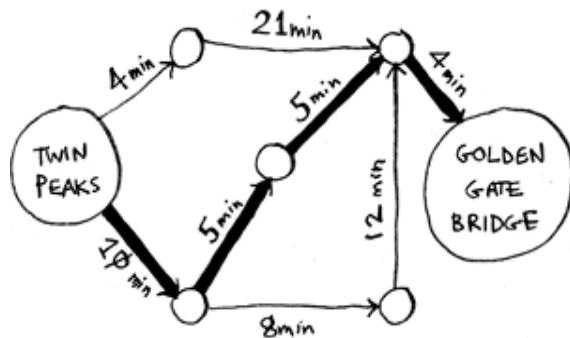
Breadth-first search wouldn't have found this as the shortest path because it has three segments. And there's a way to get from Start to Finish in two segments.

La búsqueda en amplitud no habría encontrado que este sea el camino más corto porque tiene tres segmentos. Y hay una manera de llegar del principio al fin en dos segmentos.

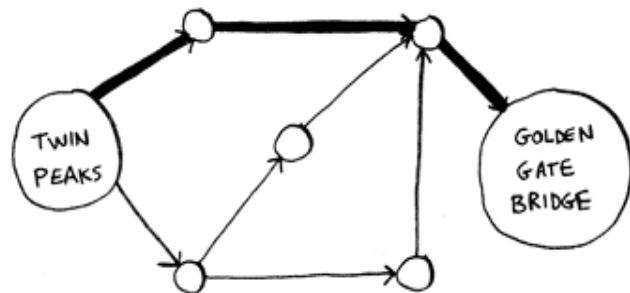


In the last chapter, you used breadth-first search to find the shortest path between two points. Back then, "shortest path" meant the path with the fewest segments. But in Dijkstra's algorithm, you assign a number or weight to each segment. Then Dijkstra's algorithm finds the path with the smallest total weight.

En el último capítulo, utilizó la búsqueda en amplitud para encontrar el camino más corto entre dos puntos. En aquel entonces, "camino más corto" significaba el camino con la menor cantidad de segmentos. Pero en el algoritmo de Dijkstra, se asigna un número o peso a cada segmento. Luego, el algoritmo de Dijkstra encuentra el camino con el peso total más pequeño.



WEIGHTED GRAPH  
(USE DIJKSTRA'S ALGORITHM)



UNWEIGHTED GRAPH  
(USE BFS)

To recap, Dijkstra's algorithm has four steps:

En resumen, el algoritmo de Dijkstra tiene cuatro pasos:

1. Find the cheapest node. This is the node you can get to in the least amount of time.

Encuentra el nodo más barato. Este es el nodo al que puedes llegar en la menor cantidad de tiempo.

2. Check whether there's a cheaper path to the out-neighbors of this node. If so, update their costs.

Compruebe si existe una ruta más barata hacia los vecinos externos de este nodo. Si es así, actualice sus costos.

3. Repeat until you've done this for every node in the graph.

Repita hasta que haya hecho esto para cada nodo del gráfico.

4. Calculate the final path. (Coming up in the next section!)

Calcula el camino final. (Próximamente en la siguiente sección!)

## Terminology

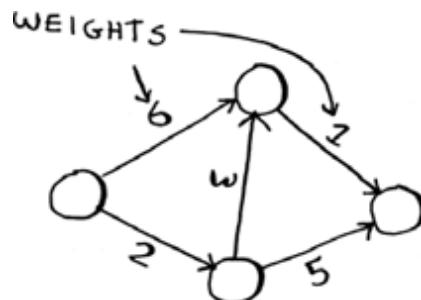
## Terminología

I want to show you some more examples of Dijkstra's algorithm in action. But, first, let me clarify some terminology.

Quiero mostrarles algunos ejemplos más del algoritmo de Dijkstra en acción. Pero primero, permítanme aclarar algo de terminología.

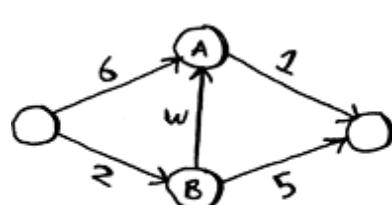
When you work with Dijkstra's algorithm, each edge in the graph has a number associated with it. These are called *weights*.

Cuando trabajas con el algoritmo de Dijkstra, cada borde del gráfico tiene un número asociado. Estos se llaman pesos.

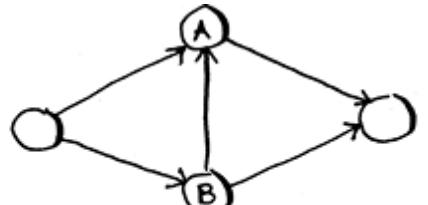


A graph with weights is called a *weighted graph*. A graph without weights is called an *unweighted graph*.

Un gráfico con ponderaciones se llama gráfico ponderado. Un gráfico sin ponderaciones se llama gráfico no ponderado.



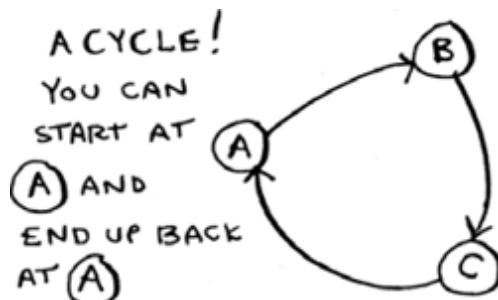
WEIGHTED GRAPH



UNWEIGHTED GRAPH

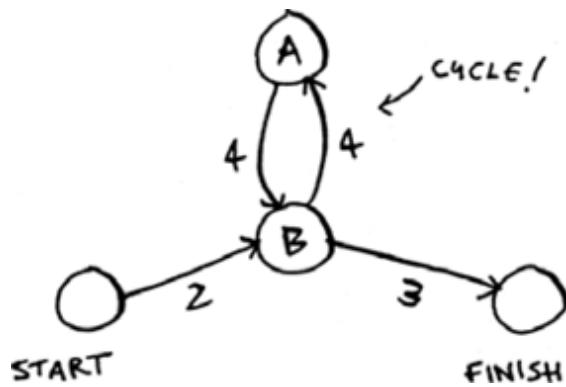
To calculate the shortest path in an unweighted graph, use *breadth-first search*. To calculate the shortest path in a weighted graph, use *Dijkstra's algorithm*. Graphs can also have *cycles*. A cycle looks like this.

Para calcular la ruta más corta en un gráfico no ponderado, utilice la búsqueda en amplitud. Para calcular el camino más corto en un gráfico ponderado, utilice el algoritmo de Dijkstra. Los gráficos también pueden tener ciclos. Un ciclo se parece a esto.



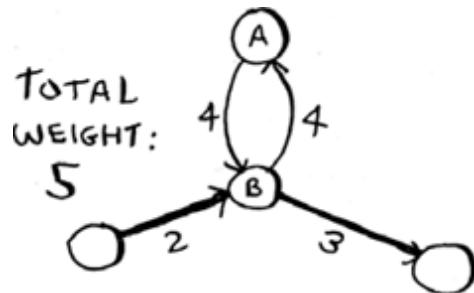
It means you can start at a node, travel around, and end up at the same node. Suppose you're trying to find the shortest path in this graph that has a cycle.

Significa que puedes comenzar en un nodo, viajar y terminar en el mismo nodo. Suponga que está intentando encontrar el camino más corto en este gráfico que tenga un ciclo.



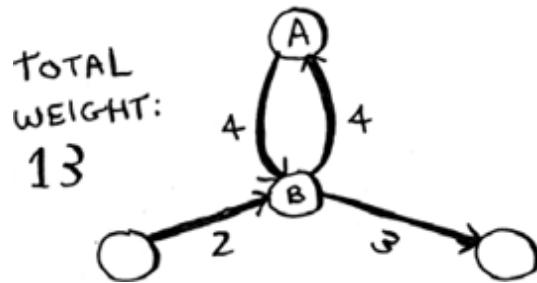
Would it make sense to follow the cycle? Well, you can use the path that avoids the cycle.

¿Tendría sentido seguir el ciclo? Bueno, puedes usar el camino que evita el ciclo.



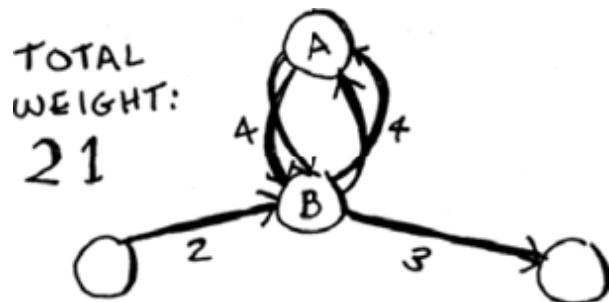
Or you can follow the cycle.

O puedes seguir el ciclo.



You end up at the target node either way, but the cycle adds more weight. You could even follow the cycle twice if you wanted.

Terminas en el nodo objetivo de cualquier manera, pero el ciclo agrega más peso. Incluso podrías seguir el ciclo dos veces si quisieras.

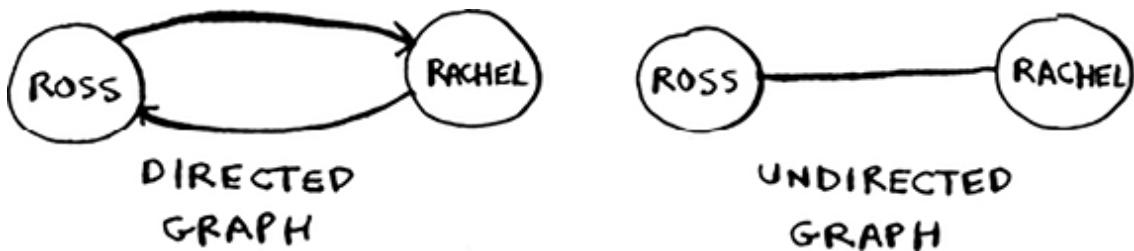


But every time you follow the cycle, you're just adding 8 to the total weight. So following the cycle will never give you the shortest path.

Pero cada vez que sigues el ciclo, solo estás sumando 8 al peso total. Así que seguir el ciclo nunca te dará el camino más corto.

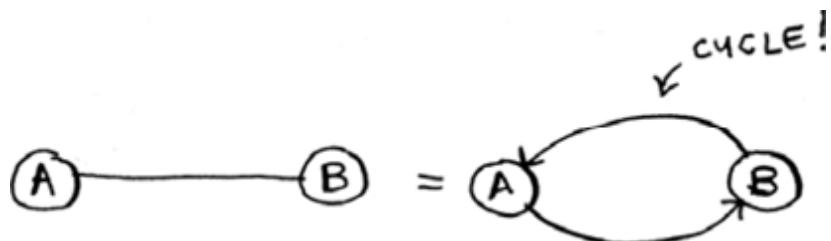
Finally, remember our conversation about directed versus undirected graphs from chapter 6?

Finalmente, ¿recuerda nuestra conversación sobre gráficos dirigidos versus no dirigidos del capítulo 6?



An undirected graph means that both nodes point to each other. That's a cycle!

Un gráfico no dirigido significa que ambos nodos apuntan entre sí. ¡Eso es un ciclo!



With an undirected graph, each edge adds another cycle. Dijkstra's algorithm only works on graphs with no cycles, where all the edges are nonnegative. Yes, it's possible for graph edges to have a negative weight! But Dijkstra's algorithm won't work—in that case, you'll need an algorithm called Bellman–Ford. There's a section on negative weight edges coming later in the chapter.

En un gráfico no dirigido, cada arista añade otro ciclo. El algoritmo de Dijkstra solo funciona en gráficos sin ciclos, donde todas las aristas son no negativas. Sí, es posible que

los bordes del gráfico tengan un peso negativo! Pero el algoritmo de Dijkstra no funcionará; en ese caso, necesitará un algoritmo llamado Bellman-Ford. Más adelante en este capítulo encontrará una sección sobre ventajas de peso negativas.

## ***Trading for a piano***

### ***Cambio por un piano***

Enough terminology, let's look at another example! This is Rama.

¡Basta de terminología, veamos otro ejemplo! Éste es Rama.



Rama is trying to trade a music book for a piano.

Rama está intentando cambiar un libro de música por un piano.

"I'll give you this poster for your book," says Alex. "It's a poster of my favorite band, Destroyer. Or I'll give you this

rare LP of Rick Astley for your book and \$5 more." "Ooh, I've heard that LP has a really great song," says Amy. "I'll trade you my guitar or drum set for the poster or the LP."

"Te daré este cartel para tu libro", dice Alex. "Es un póster de mi banda favorita, Destroyer. O te daré este raro LP de Rick Astley por tu libro y cinco dólares más". "Oh, he oído que el LP tiene una canción realmente genial", dice Amy. "Te cambio mi guitarra o batería por el cartel o el LP".



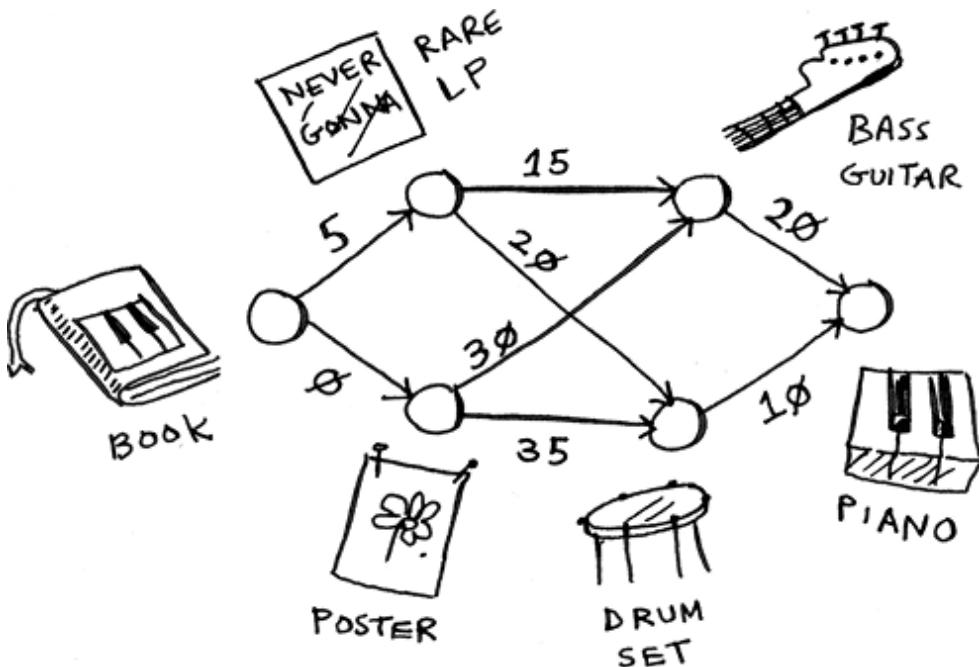
"I've been meaning to get into guitar!" exclaims Beethoven. "Hey, I'll trade you my piano for either of Amy's things."

"¡Tenía la intención de tocar la guitarra!" exclama Beethoven. "Oye, te cambio mi piano por cualquiera de las cosas de Amy".

Perfect! With a little bit of money, Rama can trade his way from a piano book to a real piano. Now he just needs to figure out how to spend the least amount of money to make those trades. Let's graph out what he's been offered.

¡Perfecto! Con un poco de dinero, Rama puede pasar de un libro de piano a un piano real. Ahora sólo necesita descubrir

cómo gastar la menor cantidad de dinero para realizar esas operaciones. Grafiquemos lo que le han ofrecido.



In this graph, the nodes are all the items Rama can trade for. The weights on the edges are the amount of money he would have to pay to make the trade. So he can trade the poster for the guitar for \$30 or trade the LP for the guitar for \$15. How is Rama going to figure out the path from the book to the piano where he spends the least dough? Dijkstra's algorithm to the rescue! Remember, Dijkstra's algorithm has four steps. In this example, you'll do all four steps, so you'll calculate the final path at the end, too.

En este gráfico, los nodos son todos los elementos por los que Rama puede intercambiar. Los pesos en los bordes son la cantidad de dinero que tendría que pagar para realizar el intercambio. Entonces puede cambiar el cartel por la guitarra por \$30 o cambiar el LP por la guitarra por \$15.

¿Cómo va a descubrir Rama el camino desde el libro hasta el piano, donde gasta menos dinero? ¡El algoritmo de Dijkstra al rescate! Recuerde, el algoritmo de Dijkstra tiene cuatro pasos. En este ejemplo, realizarás los cuatro pasos, por lo que también calcularás la ruta final al final.

NODE	COST
LP	5
POSTER	Ø
GUITAR	∞
DRUMS	∞
PIANO	∞

} WE HAVEN'T  
REACHED  
THESE NODES  
FROM THE  
START YET

Before you start, you need some setup. Make a table of the cost for each node. The cost of a node is how expensive it is to get to.

Antes de comenzar, necesita cierta configuración. Haga una tabla del costo de cada nodo. El costo de un nodo es lo costoso que es llegar a él.

You'll keep updating this table as the algorithm goes on. To calculate the final path, you also need a *parent* column on this table.

Seguirá actualizando esta tabla a medida que avance el algoritmo. Para calcular la ruta final, también necesita una columna principal en esta tabla.

NODE	PARENT
LP	BOOK
POSTER	BOOK
GUITAR	—
DRUMS	—
PIANO	—

I'll show you how this column works soon. Let's start the algorithm.

Pronto les mostraré cómo funciona esta columna.  
Comencemos el algoritmo.

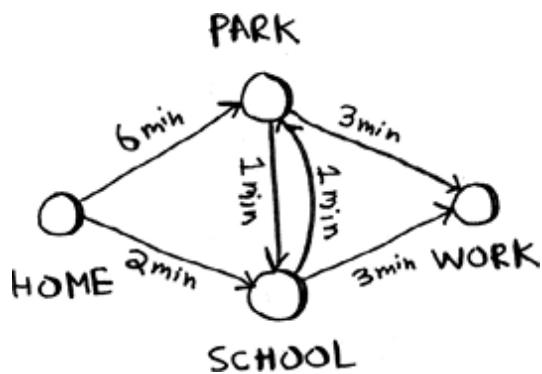
**Step 1:** Find the cheapest node. In this case, the poster is the cheapest trade at \$0. Is there a cheaper way to trade for the poster? This is a really important point, so think about it. Can you see a series of trades that will get Rama the poster for less than \$0? Read on when you're ready.

Answer: No. *Because the poster is the cheapest node Rama can get to, there's no way to make it any cheaper.* Here's a different way to look at it. Suppose you're traveling from home to work.

Paso 1: Encuentra el nodo más barato. En este caso, el cartel es el intercambio más barato a \$0. ¿Existe una forma más económica de cambiar el cartel? Este es un punto realmente importante, así que piénselo. ¿Puedes ver una serie de intercambios que le darán a Rama el

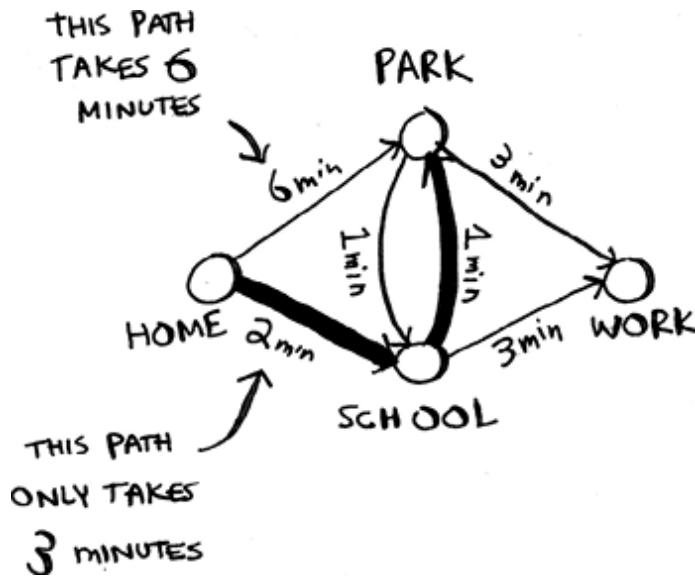
cartel por menos de \$0? Continúe leyendo cuando esté listo.

Respuesta: No. Debido a que el cartel es el nodo más barato al que Rama puede llegar, no hay manera de hacerlo más barato. Aquí hay una manera diferente de verlo. Supongamos que viaja de su casa al trabajo.



If you take the path toward the school, that takes 2 minutes. If you take the path toward the park, that takes 6 minutes. Is there any way you can take the path toward the park and end up at the school in less than 2 minutes? It's impossible because it takes longer than 2 minutes just to get to the park. On the other hand, can you find a faster path to the park? Yup.

Si tomas el camino hacia la escuela, tardarás 2 minutos. Si tomas el camino hacia el parque, tardarás 6 minutos. ¿Hay alguna manera de que puedas tomar el camino hacia el parque y terminar en la escuela en menos de 2 minutos? Es imposible porque se necesitan más de 2 minutos para llegar al parque. Por otro lado, ¿podrás encontrar un camino más rápido hacia el parque? Sí.



This is the key idea behind Dijkstra's algorithm: look at the cheapest node on your graph. There is no cheaper way to get to this node!

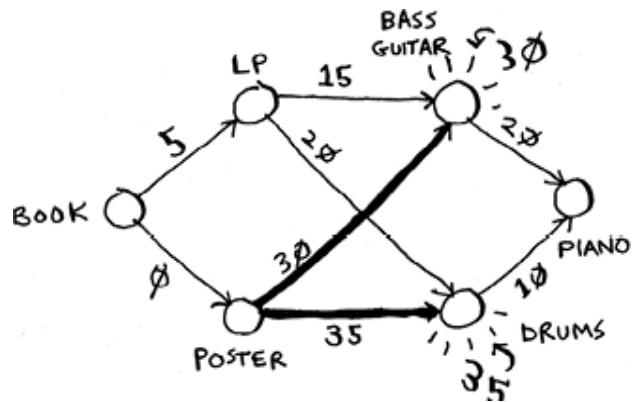
Ésta es la idea clave detrás del algoritmo de Dijkstra: mira el nodo más barato en tu gráfico. ¡No hay forma más barata de llegar a este nodo!

Back to the music example. The poster is the cheapest trade.

Volvamos al ejemplo de la música. El cartel es el comercio más barato.

**Step 2:** Figure out how long it takes to get to its out-neighbors (the cost).

Paso 2: Calcula cuánto tiempo lleva llegar a sus vecinos (el costo).



PARENT	NODE	COST
BOOK	LP	5
BOOK	POSTER	0
POSTER	GUITAR	<del>30</del>
POSTER	DRUMS	<del>35</del>
—	PIANO	$\infty$

You have prices for the bass guitar and the drum set in the table. Their value was set when you went through the poster, so the poster gets set as their parent. That means, to get to the bass guitar, you follow the edge from the poster, and the same for the drums.

Tenéis los precios del bajo y la batería en la tabla. Su valor se estableció cuando revisó el cartel, por lo que el cartel se establece como su parente. Es decir, para llegar al bajo, sigues el borde del cartel, y lo mismo para la batería.

WE GO FROM  
"POSTER" TO  
GET TO THESE  
NODES

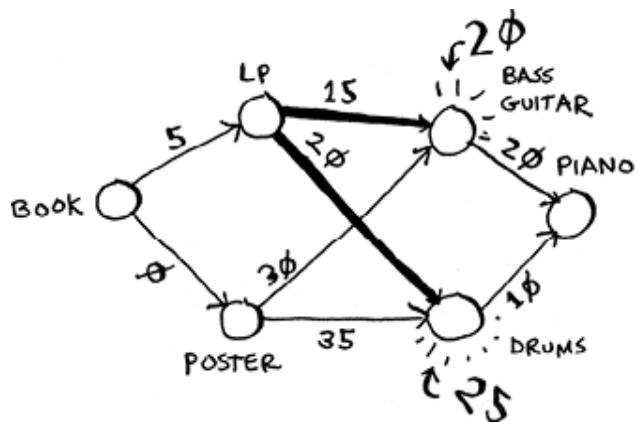
PARENT	NODE	COST
BOOK	LP	5
BOOK	POSTER	0
POSTER	GUITAR	30
POSTER	DRUMS	35
—	PIANO	$\infty$

**Step 1 again:** The LP is the next cheapest node at \$5.

Paso 1 nuevamente: El LP es el siguiente nodo más barato a \$5.

**Step 2 again:** Update the values of all of its out-neighbors.

Paso 2 nuevamente: actualice los valores de todos sus vecinos externos.



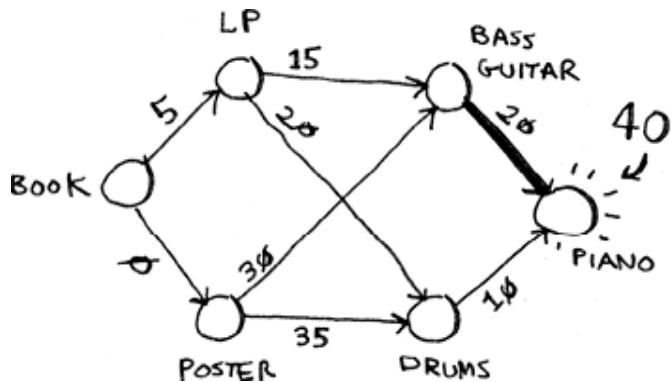
PARENT	NODE	COST
BOOK	LP	5
BOOK	POSTER	∞
LP	GUITAR	2
LP	DRUMS	35
—	PIANO	∞

Hey, you updated the price of both the drums and the guitar! That means it's cheaper to get to the drums and guitar by following the edge from the LP. So you set the LP as the new parent for both instruments.

iOye, actualizaste el precio tanto de la batería como de la guitarra! Eso significa que es más barato llegar a la batería y la guitarra siguiendo el borde del LP. Entonces configuras el LP como el nuevo padre para ambos instrumentos.

The bass guitar is the next cheapest item. Update its out-neighbors.

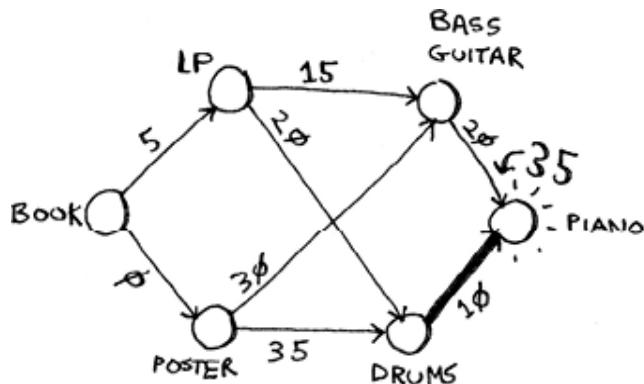
El bajo es el siguiente artículo más barato. Actualice a sus vecinos.



PARENT	NODE	COST
BOOK	LP	5
BOOK	POSTER	Ø
LP	GUITAR	2Ø
LP	DRUMS	25
GUITAR	PIANO	4Ø

OK, you finally have a price for the piano by trading the guitar for the piano. So you set the guitar as the parent. Finally, the last node, the drum set.

Bien, finalmente tienes un precio para el piano al cambiar la guitarra por el piano. Entonces configuras la guitarra como parente. Finalmente, el último nodo, la batería.



PARENT	NODE	COST
BOOK	LP	5
BOOK	POSTER	Ø
LP	GUITAR	2Ø
LP	DRUMS	25
DRUMS	PIANO	35

Rama can get the piano even cheaper by trading the drum set for the piano instead. So the cheapest set of trades will cost Rama \$35.

Rama puede conseguir el piano aún más barato cambiando la batería por el piano. Por lo tanto, el conjunto de operaciones más barato le costará a Rama 35 dólares.

Now, as I promised, you need to figure out the path. So far, you know that the shortest path costs \$35, but how do you figure out the path? To start with, look at the parent for *piano*.

Ahora, como prometí, necesitas encontrar el camino. Hasta ahora, sabes que el camino más corto cuesta \$35, pero ¿cómo calculas el camino? Para empezar, mire al padre que toca el piano.

PARENT	NODE
BOOK	LP
BOOK	POSTER
LP	GUITAR
LP	DRUMS
DRUMS	PIANO

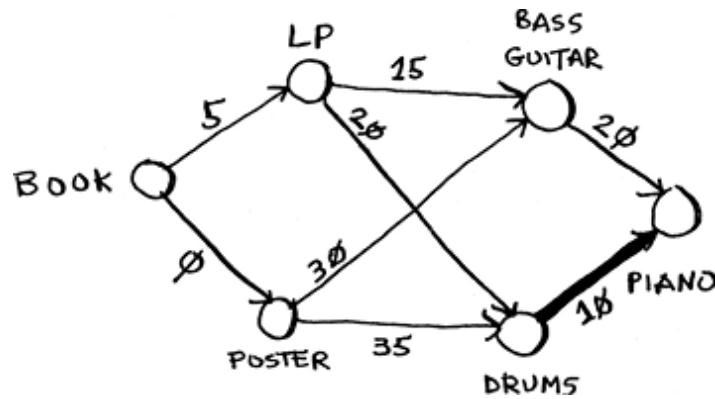
→

The piano has drums as its parent. That means Rama trades the drums for the piano. So you follow this edge.

El piano tiene la batería como parent. Eso significa que Rama cambia la batería por el piano. Entonces sigues este borde.

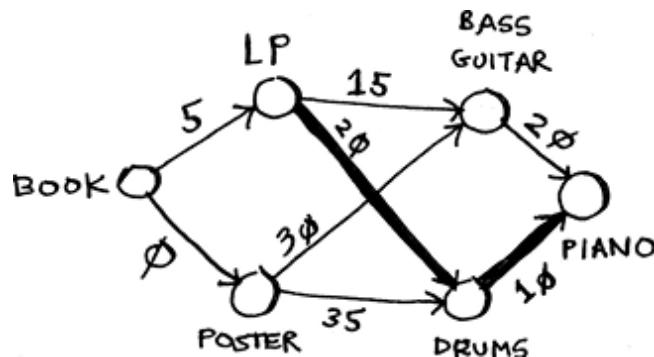
Let's see how you'd follow the edges. *Piano* has *drums* as its parent.

Veamos cómo seguirías los bordes. El piano tiene la batería como padre.



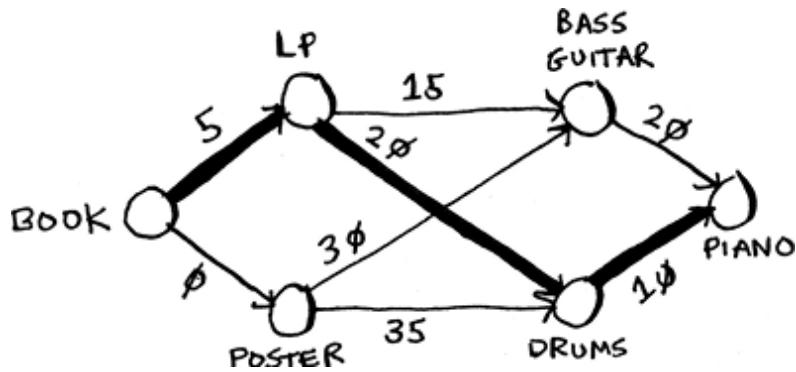
And *drums* has the LP as its parent.

Y la batería tiene al LP como parent.



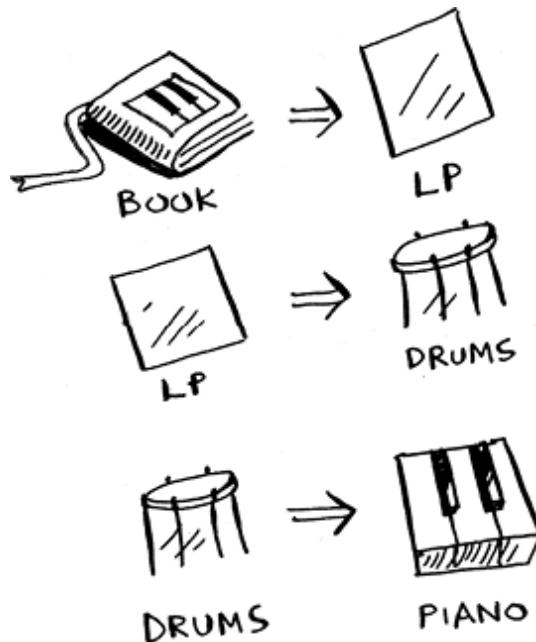
So Rama will trade the LP for the drums. And, of course, he'll trade the book for the LP. By following the parents backward, you now have the complete path.

Entonces Rama cambiará el LP por la batería. Y, por supuesto, cambiará el libro por el LP. Siguiendo a los padres hacia atrás, ahora tienes el camino completo.



Here's the series of trades Rama needs to make.

Aquí está la serie de intercambios que Rama necesita hacer.



So far, I've been using the term *shortest path* pretty literally: calculating the shortest path between two locations or between two people. I hope this example showed you that the shortest path doesn't have to be about physical distance. It can be about minimizing something. In this

case, Rama wanted to minimize the amount of money he spent. Thanks, Dijkstra!

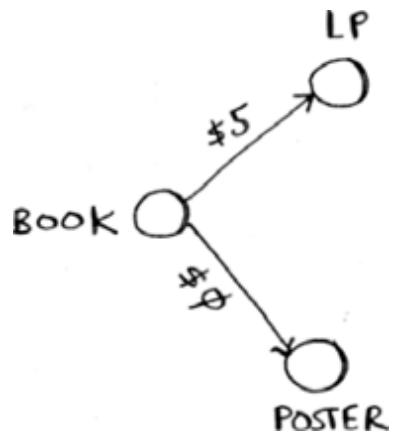
Hasta ahora, he estado usando el término camino más corto literalmente: calcular el camino más corto entre dos ubicaciones o entre dos personas. Espero que este ejemplo les haya mostrado que el camino más corto no tiene por qué tener que ver con la distancia física. Puede tratarse de minimizar algo. En este caso, Rama quería minimizar la cantidad de dinero que gastaba. ¡Gracias Dijkstra!

## **Negative-weight edges**

### **Bordes de peso negativo**

In the trading example, Alex offered to trade the book for two items.

En el ejemplo comercial, Alex se ofreció a cambiar el libro por dos artículos.

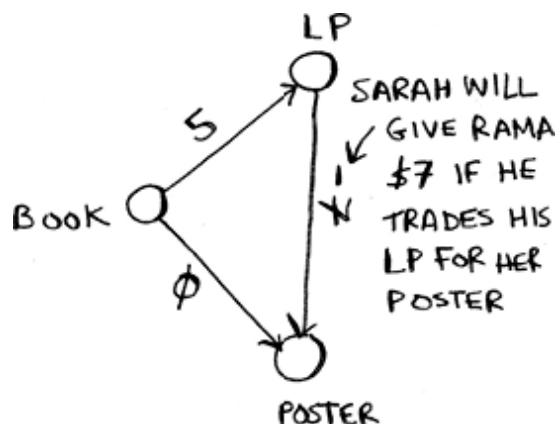


Suppose Sarah offers to trade the LP for the poster, and *she'll give Rama an additional \$7*. It doesn't cost Rama anything to make this trade; instead, he gets \$7 back.

Supongamos que Sarah se ofrece a cambiar el LP por el póster y le da a Rama \$7 adicionales. A Rama no le cuesta nada realizar este intercambio; en cambio, le devuelven \$7.

How would you show this on the graph?

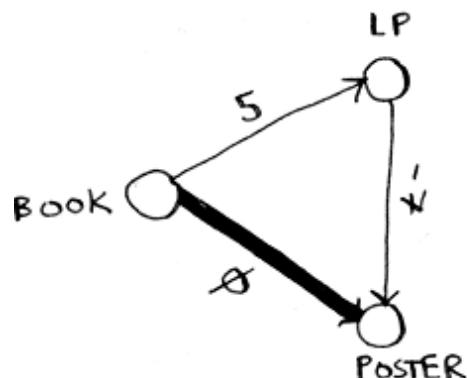
¿Cómo mostrarías esto en el gráfico?



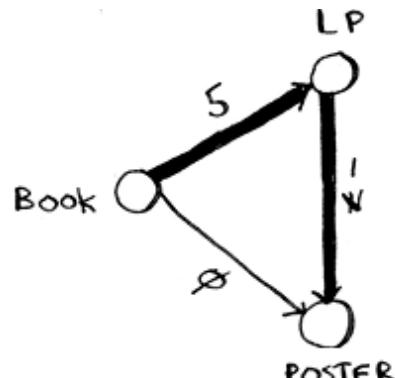
The edge from the LP to the poster has a negative weight! Rama gets \$7 back if he makes that trade. Now Rama has

two ways to get to the poster.

El borde del LP al cartel tiene un peso negativo! Rama recupera \$7 si realiza ese intercambio. Ahora Rama tiene dos maneras de llegar al cartel.



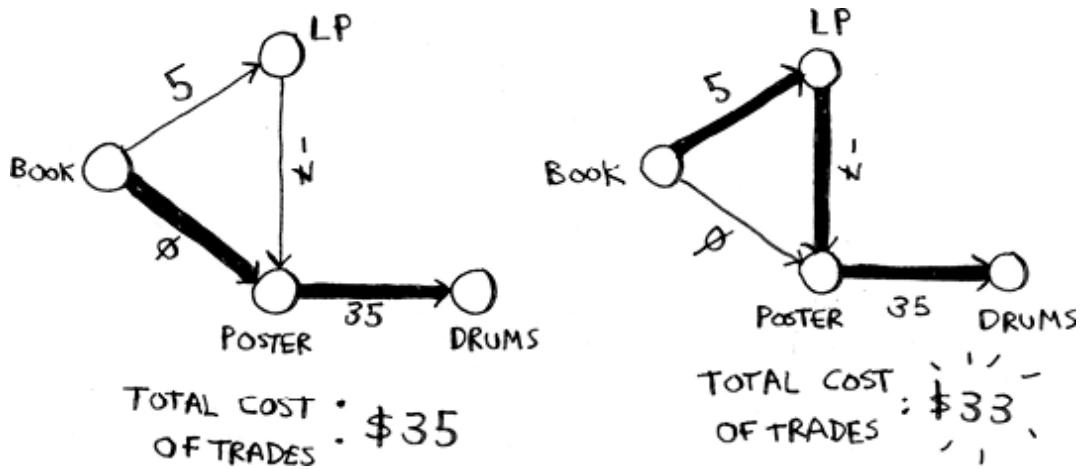
RAMA GETS \$~~0~~ BACK  
IF HE FOLLOWS  
THIS PATH



RAMA GETS \$2 BACK  
IF HE FOLLOWS THIS  
PATH

So it makes sense to do the second trade—Rama gets \$2 back that way! Now, if you remember, Rama can trade the poster for the drums. There are two paths he could take.

Así que tiene sentido hacer la segunda operación: Rama recupera \$2 de esa manera! Ahora, si recuerdas, Rama puede cambiar el cartel por los tambores. Hay dos caminos que podría tomar.



The second path costs him \$2 less, so he should take that path, right? Well, guess what? If you run Dijkstra's algorithm on this graph, Rama will take the wrong path. He'll take the longer path. *You can't use Dijkstra's algorithm if you have negative-weight edges.* Negative-weight edges break the algorithm. Let's see what happens when you run Dijkstra's algorithm on this. First, make the table of costs.

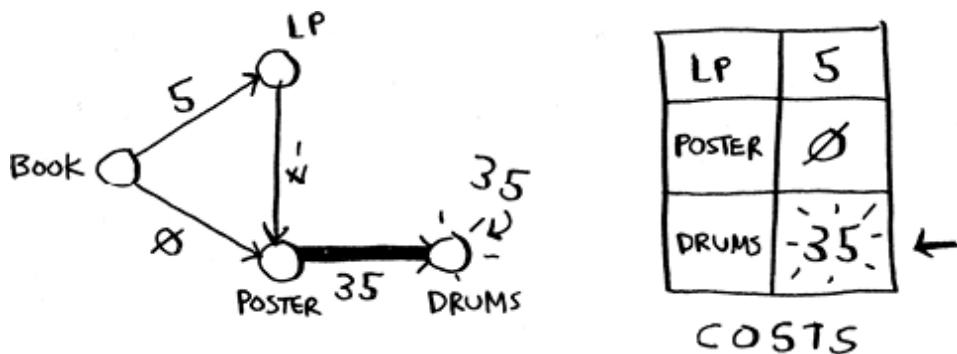
El segundo camino le cuesta \$2 menos, así que debería tomar ese camino, ¿verdad? ¿Bien adivina que? Si ejecuta el algoritmo de Dijkstra en este gráfico, Rama tomará el camino equivocado. Tomará el camino más largo. No puedes usar el algoritmo de Dijkstra si tienes ventajas de peso negativo. Los bordes de peso negativo rompen el algoritmo. Veamos qué sucede cuando ejecutas el algoritmo de Dijkstra en esto. Primero, haz la tabla de costos.

LP	5
POSTER	0
DRUMS	$\infty$

COSTS

Next, find the lowest-cost node and update the costs for its out-neighbors. In this case, the poster is the lowest-cost node. So, according to Dijkstra's algorithm, *there is no cheaper way to get to the poster than paying \$0* (you know that's wrong!). Anyway, let's update the costs for its out-neighbors.

A continuación, busque el nodo de menor costo y actualice los costos de sus vecinos. En este caso, el cartel es el nodo de menor coste. Entonces, según el algoritmo de Dijkstra, no hay forma más barata de llegar al cartel que pagar \$0 (isabes que eso está mal!). De todos modos, actualicemos los costos para sus vecinos externos.

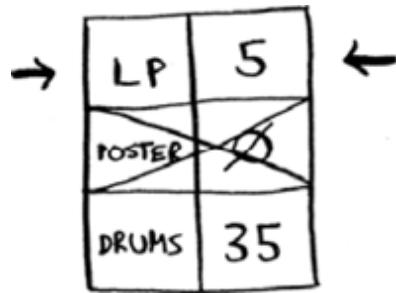


OK, the drums have a cost of \$35 now.

Ok, los tambores tienen un costo de \$35 ahora.

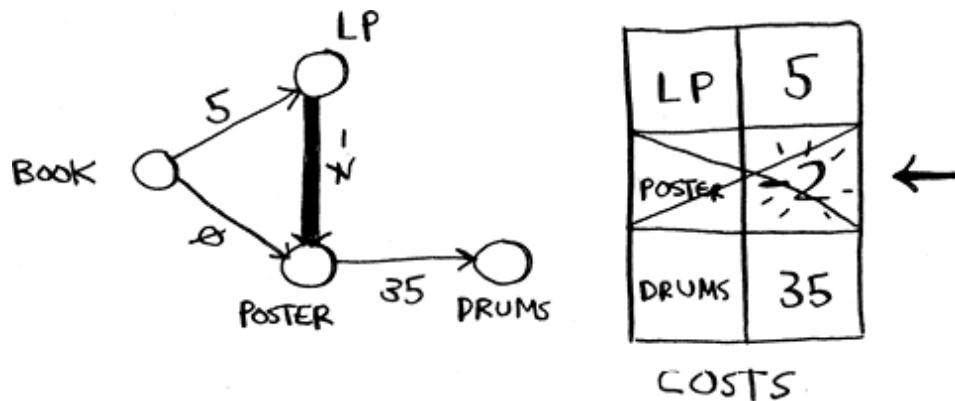
Let's get the next-cheapest node that hasn't already been processed.

Consigamos el siguiente nodo más barato que aún no haya sido procesado.



Update the costs for its out-neighbors.

Actualizar los costos para sus vecinos externos.



You already processed the poster node, but you're updating the cost for it. This is a big red flag. Once you process a node, it means there's no cheaper way to get to that node. But you just found a cheaper way to the poster! Drums doesn't have any out-neighbors, so that's the end of the algorithm. Here are the final costs.

Ya procesaste el nodo del póster, pero estás actualizando el costo del mismo. Esta es una gran señal de alerta. Una vez que procesas un nodo, significa que no hay forma más barata de llegar a ese nodo. ¡Pero acabas de encontrar una forma más económica de acceder al póster! Drums no tiene vecinos, así que ese es el final del algoritmo. Aquí están los costos finales.

LP	5
POSTER	-2
DRUMS	35

FINAL  
COSTS

It costs \$35 to get to the drums. You know that there's a path that costs only \$33, but Dijkstra's algorithm didn't find it. Dijkstra's algorithm assumed that because you were processing the poster node, there was no cheaper way to get to that node. That assumption only works if you have no negative-weight edges. So you *can't use negative-weight edges with Dijkstra's algorithm*. If you want to find the shortest path in a graph that has negative-weight edges, there's an algorithm for that! It's called the *Bellman–Ford algorithm*. Bellman–Ford is out of the scope of this book, but you can find some great explanations online.

Cuesta \$35 llegar a la batería. Sabes que hay un camino que cuesta sólo \$33, pero el algoritmo de Dijkstra no lo

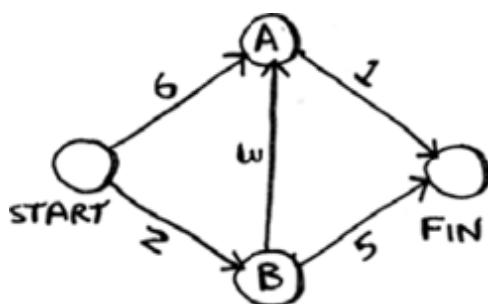
encontró. El algoritmo de Dijkstra suponía que, dado que se estaba procesando el nodo del cartel, no había una forma más barata de llegar a ese nodo. Esa suposición sólo funciona si no tienes ventajas de peso negativo. Por lo tanto, no se pueden utilizar aristas de peso negativo con el algoritmo de Dijkstra. Si desea encontrar el camino más corto en un gráfico que tiene aristas de peso negativo, ¡existe un algoritmo para eso! Se llama algoritmo de Bellman-Ford. Bellman-Ford está fuera del alcance de este libro, pero puede encontrar excelentes explicaciones en línea.

## ***Implementation***

## ***Implementación***

Let's see how to implement Dijkstra's algorithm in code. Here's the graph I'll use for the example.

Veamos cómo implementar el algoritmo de Dijkstra en código. Aquí está el gráfico que usaré para el ejemplo.



To code this example, you'll need three hash tables.

Para codificar este ejemplo, necesitará tres tablas hash.

START	A	6
	B	2
A	FIN	1
B	A	3
	FIN	5
FIN		—

GRAPH

A	6
B	2
FIN	$\infty$

COSTS

A	START
B	START
FIN	—

PARENTS

You'll update the costs and parents hash tables as the algorithm progresses. First, you need to implement the graph. You'll use a hash table like you did in chapter 6:

Actualizará las tablas hash de costos y padres a medida que avance el algoritmo. Primero, necesitas implementar el gráfico. Utilizarás una tabla hash como lo hiciste en el capítulo 6:

```
graph = {}
```

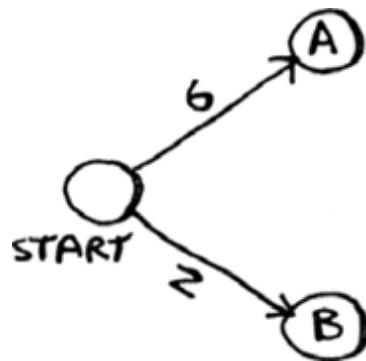
In the last chapter, you stored all the out-neighbors of a node in the hash table, like this:

En el último capítulo, almacenaste todos los vecinos externos de un nodo en la tabla hash, así:

```
graph["you"] = ["alice", "bob", "claire"]
```

But this time, you need to store the out-neighbors *and* the cost for getting to that neighbor. For example, Start has two out-neighbors, A and B.

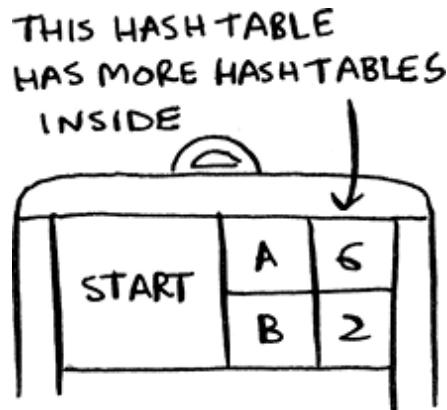
Pero esta vez, necesita almacenar a los vecinos externos y el costo de llegar a ese vecino. Por ejemplo, Start tiene dos vecinos externos, A y B.



How do you represent the weights of those edges? Why not just use another hash table?

¿Cómo representas los pesos de esos bordes? ¿Por qué no utilizar otra tabla hash?

```
graph["start"] = {}  
graph["start"]["a"] = 6  
graph["start"]["b"] = 2
```



So `graph["start"]` is a hash table. You can get all the out-neighbors for Start like this:

Entonces `graph["start"]` es una tabla hash. Puede obtener todos los vecinos externos para comenzar de esta manera:

```
>>> print(list(graph["start"].keys()))
["a", "b"]
```

There's an edge from Start to A and an edge from Start to B. What if you want to find the weights of those edges?

Hay una arista de Inicio a A y una arista de Inicio a B. ¿Qué pasa si quieres encontrar los pesos de esas aristas?

```
>>> print(graph["start"]["a"])
6
>>> print(graph["start"]["b"])
2
```

Let's add the rest of the nodes and their out-neighbors to the graph:

Agreguemos el resto de los nodos y sus vecinos al gráfico:

```

graph["a"] = {}
graph["a"]["fin"] = 1

graph["b"] = {}
graph["b"]["a"] = 3
graph["b"]["fin"] = 5

graph["fin"] = {} ①

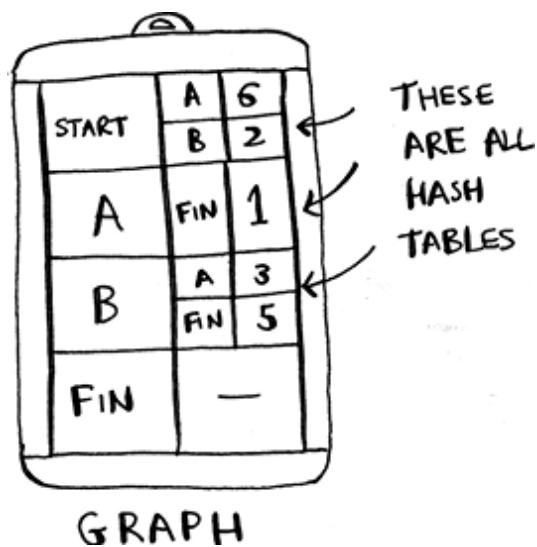
```

① The Finish node doesn't have any out-neighbors.

① El nodo Finalizar no tiene vecinos externos.

The full graph hash table looks like this.

La tabla hash del gráfico completo se ve así.



Next, you need a hash table to store the current costs for each node.

A continuación, necesita una tabla hash para almacenar los costos actuales de cada nodo.

A	6
B	2
FIN	∞

COSTS

The *cost* of a node is how long it takes to get to that node from Start. You know it takes 2 minutes from Start to node B. You know it takes 6 minutes to get to node A (although you may find a path that takes less time). You don't know how long it takes to get to Finish. If you don't know the cost yet, you put down infinity. Can you represent *infinity* in Python? Turns out, you can:

El costo de un nodo es el tiempo que lleva llegar a ese nodo desde el inicio. Sabes que se necesitan 2 minutos desde Inicio hasta el nodo B. Sabes que se necesitan 6 minutos para llegar al nodo A (aunque es posible que encuentres un camino que demore menos tiempo). No sabes cuánto tiempo lleva llegar a Finalizar. Si aún no sabes el costo, anotas infinito. ¿Puedes representar el infinito en Python? Resulta que puedes:

```
infinity = math.inf
```

Here's the code to make the costs table:

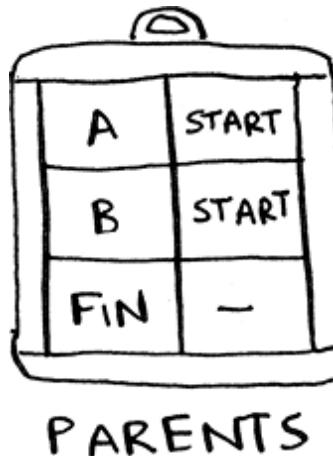
Aquí está el código para hacer la tabla de costos:

```
infinity = math.inf
costs = {}
costs["a"] = 6
```

```
costs["b"] = 2  
costs["fin"] = infinity
```

You also need another hash table for the parents:

También necesitas otra tabla hash para los padres:



Here's the code to make the hash table for the parents:

Aquí está el código para crear la tabla hash para los padres:

```
parents = {}  
parents["a"] = "start"  
parents["b"] = "start"  
parents["fin"] = None
```

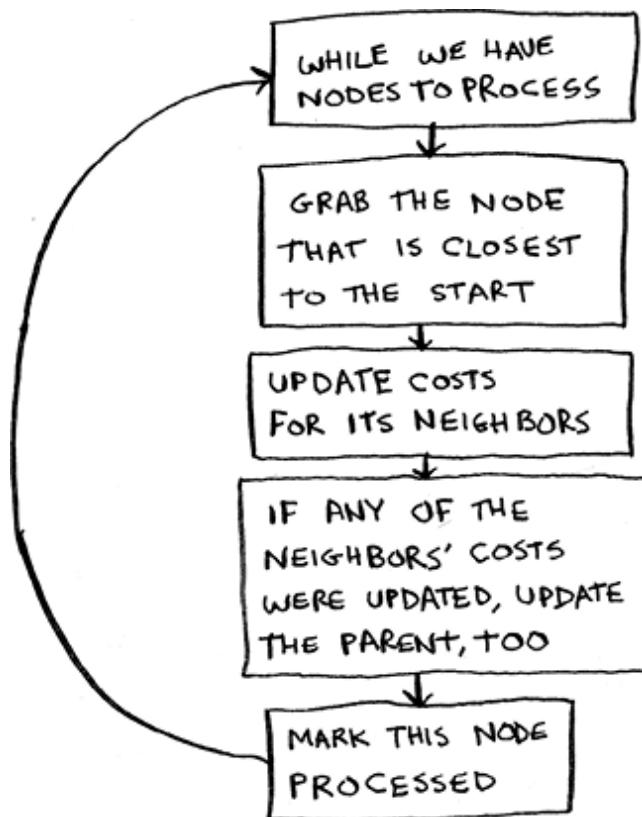
Finally, you need a set to keep track of all the nodes you've already processed because you don't need to process a node more than once:

Finalmente, necesita un conjunto para realizar un seguimiento de todos los nodos que ya procesó porque no necesita procesar un nodo más de una vez:

```
processed = set()
```

That's all the setup. Now let's look at the algorithm.

Esa es toda la configuración. Ahora veamos el algoritmo.



I'll show you the code first and then walk through it. Here's the code:

Primero te mostraré el código y luego lo revisaré. Aquí está el código:

```
node = find_lowest_cost_node(costs)      ①
while node is not None:                  ②
    cost = costs[node]
    neighbors = graph[node]
    for n in neighbors.keys():            ③
```

```

new_cost = cost + neighbors[n]
if costs[n] > new_cost:          ④
    costs[n] = new_cost          ⑤
    parents[n] = node           ⑥
processed.add(node)              ⑦
node = find_lowest_cost_node(costs) ⑧

```

- ① Finds the lowest-cost node that you haven't processed yet
- ① Encuentra el nodo de menor costo que aún no ha procesado
- ② If you've processed all the nodes, this while loop is done.
- ② Si ha procesado todos los nodos, este ciclo while estará completo.
- ③ Goes through all the out-neighbors of this node
- ③ Pasa por todos los vecinos externos de este nodo
- ④ If it's cheaper to get to this out-neighbor by going through this node . . .
- ④ Si es más barato llegar a este vecino pasando por este nodo. . .
- ⑤ . . . updates the cost for the neighbor
- ⑤ . . . actualiza el costo para el vecino
- ⑥ This node becomes the new parent for this out-neighbor.
- ⑥ Este nodo se convierte en el nuevo parente de este vecino externo.
- ⑦ Marks the node as processed
- ⑦ Marca el nodo como procesado
- ⑧ Finds the next node to process and loops
- ⑧ Encuentra el siguiente nodo para procesar y realiza un bucle.

That's Dijkstra's algorithm in Python! Let's see this code in action. Then I'll show you the code for the `find_lowest_cost_node` function.

Ese es el algoritmo de Dijkstra en Python! Veamos este código en acción. Luego te mostraré el código de la función `find_lowest_cost_node`.

Find the node with the lowest cost.

Encuentre el nodo con el menor costo.

*NODE IS "B"* → `node = find_lowest_cost_node(costs)` →

A	6
B	2
FIN	∞

costs

Get the cost and out-neighbors of that node.

Obtenga el costo y los vecinos de ese nodo.

*cost is 2* → `cost = costs[node]`

*neighbors* → `neighbors = graph[node]`

*neighbors is a hash table:*

A	3
FIN	5

→

START	A	6
	B	2
A	FIN	1
B	A	∞
	FIN	5
FIN	—	

GRAPH

Loop through the out-neighbors.

Recorra a los vecinos.

`for n in neighbors.keys():`

*n is "A"*

*A LIST OF NODES:*

A	FIN
---	-----

KEYS    VALUES

A	3
FIN	5

Each node has a cost. The cost is how long it takes to get to that node from Start. Here, you're calculating how long it

would take to get to node A if you went Start > node B > node A, instead of Start > node A.

Cada nodo tiene un costo. El costo es el tiempo que lleva llegar a ese nodo desde el inicio. Aquí, estás calculando cuánto tiempo tomaría llegar al nodo A si fueras Inicio > nodo B > nodo A, en lugar de Inicio > nodo A.

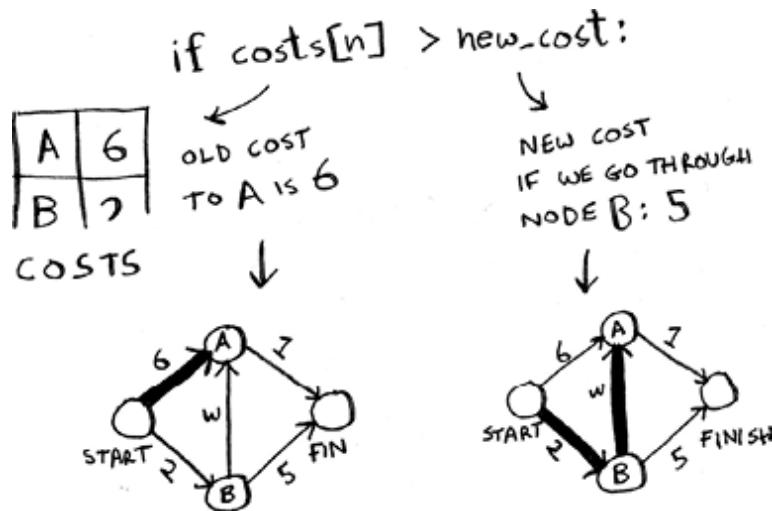
$$\text{new\_cost} = \text{cost} + \text{neighbors}[n]$$

↑                              ↓  
 COST OF                    DISTANCE FROM  
 "B", i.e. 2                B TO A: 3

} new\\_cost = 2 + 3  
 = 5

Let's compare those costs.

Comparemos esos costos.



You found a shorter path to node A! Update the cost.

¡Encontraste un camino más corto al nodo A! Actualizar el costo.

$\text{costs}[n] = \text{new\_cost}$

↑      ↑  
"A"    5

A	5
B	2
FIN	$\infty$

COSTS

The new path goes through node B, so set B as the new parent.

La nueva ruta pasa por el nodo B, así que establezca B como nuevo parente.

$\text{parents}[n] = \text{node}$

↑      ↑  
"A"    "B"

A	B
B	START
FIN	-

PARENTS

OK, you're back at the top of the loop. The next out-neighbor in the `for` loop is the Finish node.

Bien, estás de vuelta en la parte superior del bucle. El siguiente vecino externo en el bucle `for` es el nodo Finalizar.

for n in neighbors.keys():  
 ↑  
 n is  
 "FIN"  
 { }  
 A | FIN

How long does it take to get to Finish if you go through node B?

¿Cuánto tiempo lleva llegar a Finalizar si pasas por el nodo B?

$$\text{new\_cost} = \text{cost} + \text{neighbors}[h]$$

↓                              ↓  
 2                              DISTANCE FROM  
 B TO THE FINISH:  
 5

}  
 2 + 5  
 = 7

It takes 7 minutes. The previous cost was infinity minutes, and 7 minutes is less than that.

Tarda 7 minutos. El coste anterior era infinitos minutos y 7 minutos es menos que eso.

		$\text{if } \text{costs}[n] > \text{new\_cost}:$
FIN	$\infty$	<p>WE HAD NO COST TO THE FINISH BEFORE THIS</p>

Set the new cost and the new parent for the Finish node.

Establezca el nuevo costo y el nuevo parent para el nodo Finalizar.

$$\text{costs}[n] = \text{new\_cost}$$

↑      ↑  
"FIN"    7

A	5
B	2
FIN	???

←  
COSTS

$$\text{parents}[n] = \text{node}$$

↑      ↑  
"FIN"    "B"

A	B
B	START
FIN	???

←  
PARENTS

OK, you updated the costs for all the out-neighbors of node B. Mark it as processed.

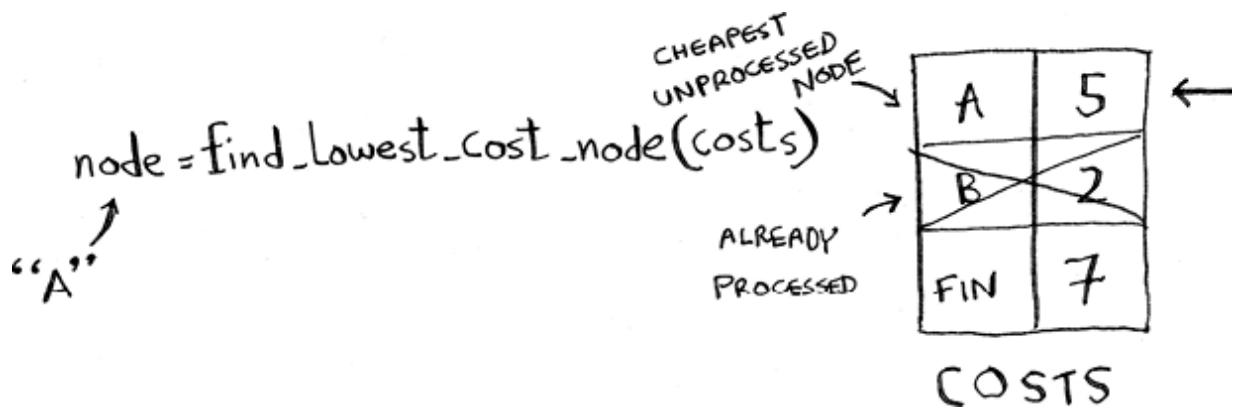
Bien, actualizó los costos de todos los vecinos externos del nodo B. Márquelo como procesado.

processed.append(node)  
" " B" " ↑

PROCESSED  
NODES: B

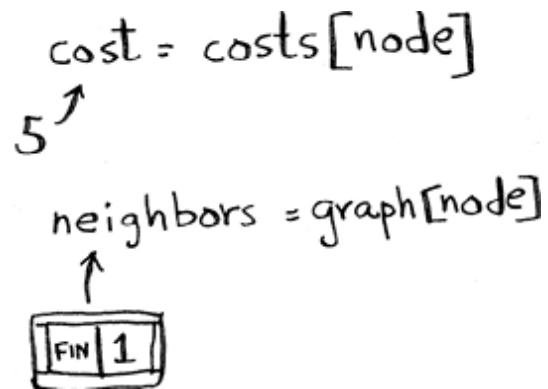
Find the next node to process.

Encuentre el siguiente nodo para procesar.



Get the cost and out-neighbors for node A.

Obtenga el costo y los vecinos externos del nodo A.



Node A only has one out-neighbor: the Finish node.

El nodo A solo tiene un vecino externo: el nodo Finalizar.



Currently, it takes 7 minutes to get to the Finish node. How long would it take to get there if you went through node A?

Actualmente, se necesitan 7 minutos para llegar al nodo Finalizar. ¿Cuánto tiempo tardaría en llegar si pasara por el nodo A?

$$\text{new\_cost} = \text{cost} + \text{neighbors}[n]$$

↓                      ↓  
 COST TO            DISTANCE FROM  
 GET TO A            A TO THE FINISH:  
 FROM THE            1  
 START: 5

} = 6

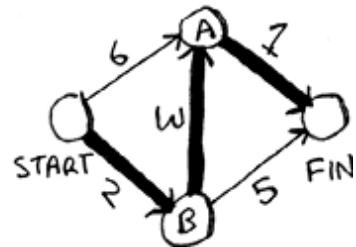
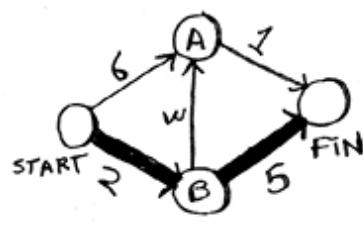
if  $\text{costs}[n] > \text{new\_cost}$ :

D	2
FIN	7

COSTS

↓  
 OLD COST  
 TO GET TO  
 THE FINISH: 7

↓  
 COST IF WE  
 GO THROUGH A:  
 6



It's faster to get to Finish from node A! Let's update the cost and parent.

¡Es más rápido llegar a Finalizar desde el nodo A!  
 Actualicemos el costo y el parent.

$\text{costs}[n] = \text{new\_cost}$

↑      ↑  
 "FIN"    6

A	5
B	2
FIN	6

COSTS

$\text{parents}[n] = \text{node}$

↑      ↑  
 "FIN"    "A"

A	B
B	START
FIN	A

PARENTS

Once you've processed all the nodes, the algorithm is over. I hope the walkthrough helped you understand the algorithm a little better. Finding the lowest-cost node is pretty easy with the `find_lowest_cost_node` function. Here it is in code:

Una vez que haya procesado todos los nodos, el algoritmo habrá terminado. Espero que el tutorial te haya ayudado a comprender un poco mejor el algoritmo. Encontrar el nodo de menor costo es bastante fácil con la función `find_lowest_cost_node`. Aquí está en código:

```
def find_lowest_cost_node(costs):
    lowest_cost = math.inf
    lowest_cost_node = None
    for node in costs: ①
        cost = costs[node]
        if cost < lowest_cost and node not in processed: ②
            lowest_cost = cost ③
            lowest_cost_node = node
    return lowest_cost_node
```

- ① Goes through each node
- ① Pasa por cada nodo
- ② If it's the lowest cost so far and hasn't been processed yet . . .
- ② Si es el costo más bajo hasta el momento y aún no se ha procesado. . .
- ③ . . . sets it as the new lowest-cost node
- ③ . . . lo establece como el nuevo nodo de menor costo

To find the lowest cost node, we loop through all the nodes each time. There is a more efficient version of this algorithm. It uses a data structure called a priority queue. A priority queue is itself built on top of a different data structure called a heap. If you're curious about priority queues and heaps, check out the section on heaps in the last chapter of the book.

Para encontrar el nodo de menor costo, recorremos todos los nodos cada vez. Existe una versión más eficiente de este algoritmo. Utiliza una estructura de datos llamada cola de prioridad. Una cola de prioridad se construye sobre una estructura de datos diferente llamada montón. Si tiene curiosidad acerca de las colas y los montones de prioridad, consulte la sección sobre montones en el último capítulo del libro.

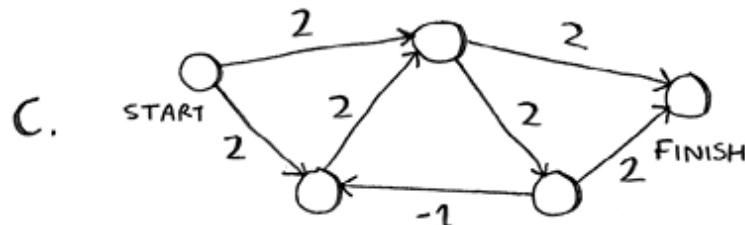
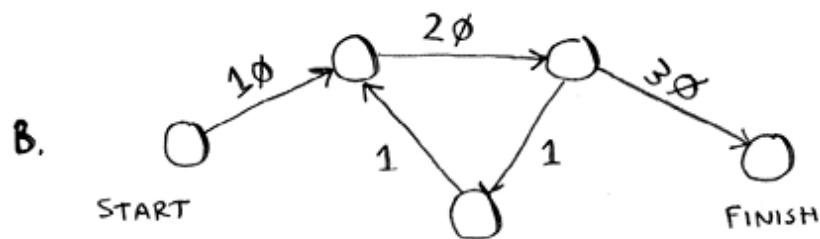
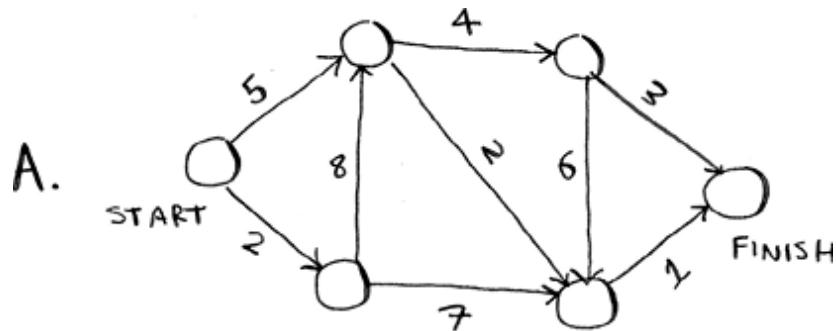
## EXERCISE

### EJERCICIO

**9.1** In each of these graphs, what is the weight of the shortest path from Start to Finish?

9.1 En cada uno de estos gráficos, ¿cuál es el peso del camino más corto desde el principio hasta el

final?



## Recap

## Resumen

- Breadth-first search is used to calculate the shortest path for an unweighted graph.  
La búsqueda en amplitud se utiliza para calcular la ruta más corta para un gráfico no ponderado.

- Dijkstra's algorithm is used to calculate the shortest path for a weighted graph.

El algoritmo de Dijkstra se utiliza para calcular el camino más corto para un gráfico ponderado.

- Dijkstra's algorithm works when all the weights are nonnegative.

El algoritmo de Dijkstra funciona cuando todos los pesos no son negativos.

- If you have negative weights, use the Bellman–Ford algorithm.

Si tiene pesos negativos, utilice el algoritmo de Bellman–Ford.

# **10 Greedy algorithms**

---

## **10 algoritmos codiciosos**

---

### **In this chapter**

#### **En este capítulo**

- You learn about the greedy strategy, a very simple problem-solving strategy.  
Aprendes sobre la estrategia codiciosa, una estrategia muy simple para resolver problemas.
- You learn how to cope with the impossible: problems that have no fast algorithmic solution (NP-hard problems).  
Aprendes a afrontar lo imposible: problemas que no tienen una solución algorítmica rápida (problemas NP-difíciles).
- You learn about approximation algorithms, which you can use to find an approximate solution to an NP-hard problem quickly.  
Aprenderá sobre algoritmos de aproximación, que puede utilizar para encontrar rápidamente una solución aproximada a un problema NP-difícil.
- You learn about the greedy strategy, a very simple problem-solving strategy.  
Aprendes sobre la estrategia codiciosa, una estrategia muy simple para resolver problemas.

## ***The classroom scheduling problem***

### ***El problema de la programación del aula***

Suppose you have a classroom and want to hold as many classes here as possible. You get a list of classes.

Supongamos que tiene un salón de clases y desea realizar tantas clases aquí como sea posible. Obtienes una lista de clases.



You can't hold *all* of these classes in there because some of them overlap.

No puedes impartir todas estas clases allí porque algunas se superponen.

# **11 Dynamic programming**

---

## **11 Programación dinámica**

---

### **In this chapter**

#### **En este capítulo**

- You learn dynamic programming, a technique to solve a hard problem by breaking it up into subproblems and solving those subproblems first.  
Aprende programación dinámica, una técnica para resolver un problema difícil dividiéndolo en subproblemas y resolviendo esos subproblemas primero.
- Using examples, you learn how to come up with a dynamic programming solution to a new problem.  
Usando ejemplos, aprenderá cómo encontrar una solución de programación dinámica para un nuevo problema.

### ***The knapsack problem (revisited)***

### ***El problema de la mochila (revisado)***

Let's revisit the knapsack problem from chapter 10. You're a thief with a knapsack that can carry 4 lb of goods.

Repasemos el problema de la mochila del capítulo 10. Eres un ladrón con una mochila que puede transportar 4 libras de

mercancías.



You have three items that you can put into the knapsack.

Tienes tres artículos que puedes poner en la mochila.



STEREO  
\$3000  
4 lbs



LAPTOP  
\$2000  
3 lbs



GUITAR  
\$1500  
1 lbs

What items should you steal so that you steal the maximum money's worth of goods?

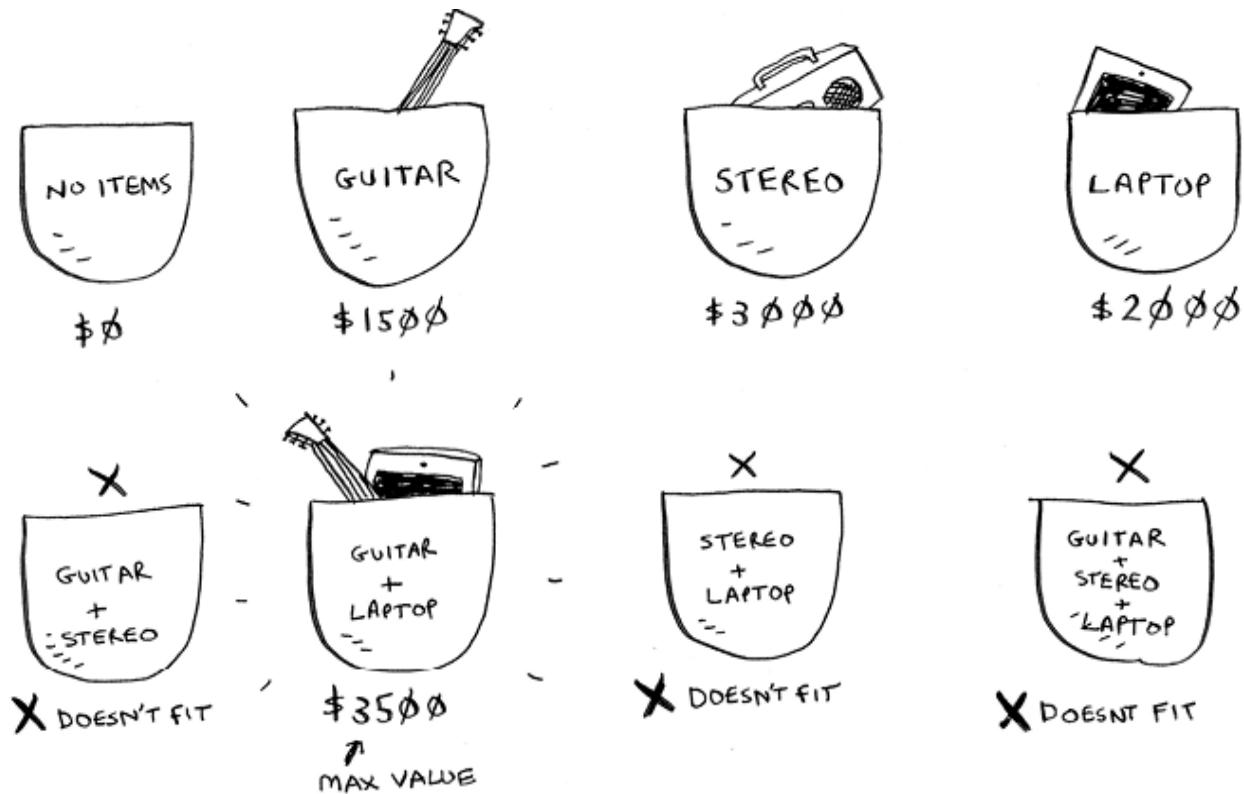
¿Qué artículos debería robar para robar el máximo valor de dinero en bienes?

## The simple solution

### La solución sencilla

The simplest algorithm is this: you try every possible set of goods and find the set that gives you the most value.

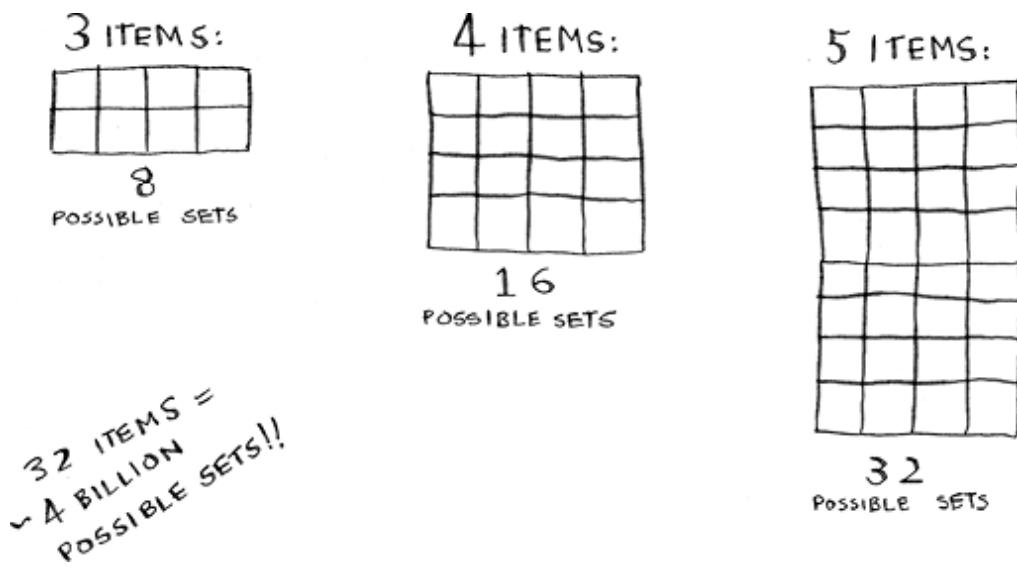
El algoritmo más simple es este: prueba todos los conjuntos posibles de bienes y encuentra el conjunto que le brinda el mayor valor.



This works, but it's really slow. For 3 items, you have to calculate 8 possible sets. For 4 items, you have to calculate 16 sets. With every item you add, the number of sets you

have to calculate doubles! This algorithm takes  $O(2n)$  time, which is very, very slow.

Esto funciona, pero es muy lento. Para 3 artículos, debes calcular 8 conjuntos posibles. Para 4 artículos, debes calcular 16 conjuntos. ¡Con cada elemento que agregas, la cantidad de conjuntos que tienes que calcular se duplica! Este algoritmo requiere un tiempo  $O(2n)$ , que es muy, muy lento.



That's impractical for any reasonable number of goods. In chapter 10, you saw how to calculate an *approximate* solution. That solution will be close to the optimal solution, but it may not be the optimal solution.

Esto no es práctico para una cantidad razonable de bienes. En el capítulo 10, viste cómo calcular una solución aproximada. Esa solución estará cerca de la solución óptima, pero puede que no sea la solución óptima.

So how do you calculate the optimal solution?

Entonces, ¿cómo se calcula la solución óptima?

## Dynamic programming

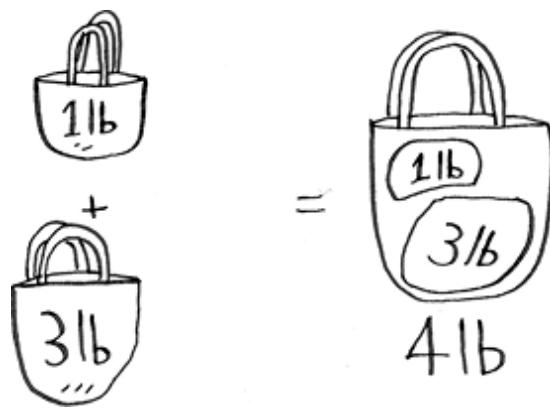
### Programación dinámica

Answer: With dynamic programming! Let's see how the dynamic programming algorithm works here. Dynamic programming starts by solving subproblems and builds up to solving the big problem.

Respuesta: ¡Con programación dinámica! Veamos cómo funciona aquí el algoritmo de programación dinámica. La programación dinámica comienza resolviendo subproblemas y continúa hasta resolver el gran problema.

For the knapsack problem, you'll start by solving the problem for smaller knapsacks (or sub-knapsacks) and then work up to solving the original problem.

Para el problema de la mochila, comenzará resolviendo el problema de las mochilas (o submochilas) más pequeñas y luego trabajará hasta resolver el problema original.



*Dynamic programming is a hard concept, so don't worry if you don't get it right away. We're going to look at a lot of examples.*

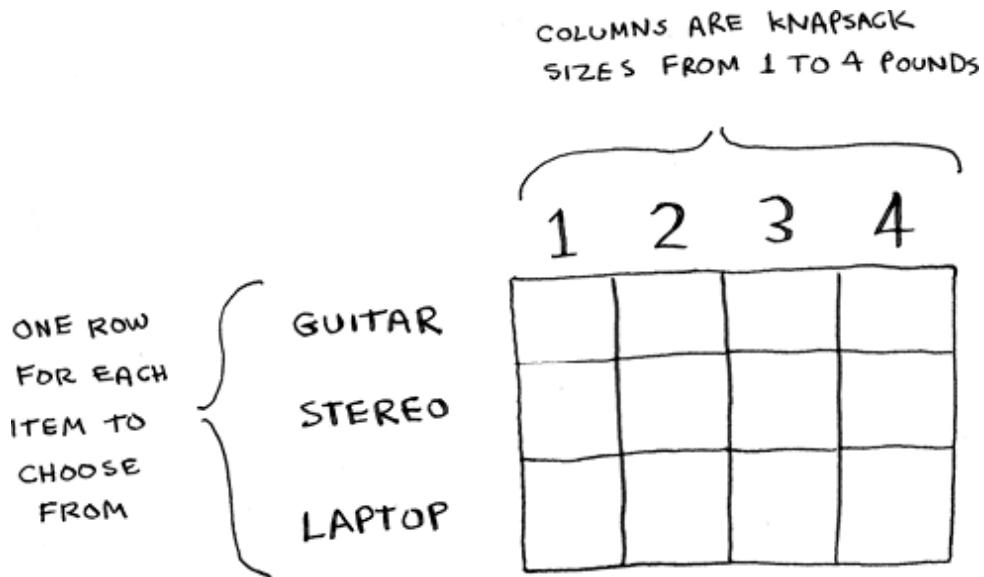
La programación dinámica es un concepto difícil, así que no se preocupe si no lo entiende de inmediato. Vamos a ver muchos ejemplos.

I'll start by showing you the algorithm in action. After you've seen it in action once, you'll have a lot of questions! I'll do my best to address every question.

Comenzaré mostrándoles el algoritmo en acción. ¡Después de haberlo visto en acción una vez, tendrás muchas preguntas! Haré todo lo posible para abordar cada pregunta.

Every dynamic programming algorithm starts with a grid. Here's a grid for the knapsack problem.

Todo algoritmo de programación dinámica comienza con una cuadrícula. Aquí hay una cuadrícula para el problema de la mochila.



The rows of the grid are the items, and the columns are knapsack weights from 1 lb to 4 lb. You need all of those columns because they will help you calculate the values of the sub-knapsacks.

Las filas de la cuadrícula son los artículos y las columnas son los pesos de las mochilas de 1 libra a 4 libras. Necesitas todas esas columnas porque te ayudarán a calcular los valores de las submochilas.

The grid starts out empty. You're going to fill in each cell of the grid. Once the grid is filled in, you'll have your answer to this problem! Please follow along. Make your own grid, and we'll fill it out together.

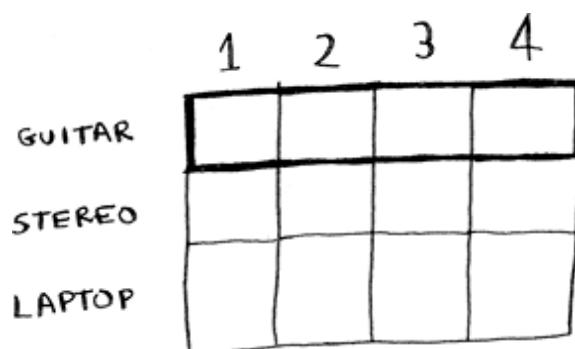
La cuadrícula comienza vacía. Vas a completar cada celda de la cuadrícula. Una vez que complete la cuadrícula, itendrá su respuesta a este problema! Por favor sigue adelante. Haz tu propia cuadrícula y la completaremos juntos.

## THE GUITAR ROW

### LA FILA DE GUITARRAS

I'll show you the exact formula for calculating this grid later. Let's do a walkthrough first. Start with the first row.

Más adelante te mostraré la fórmula exacta para calcular esta cuadrícula. Primero hagamos un recorrido. Comience con la primera fila.



This is the *guitar* row, which means you're trying to fit the guitar into the knapsack. At each cell, there's a simple decision: Do you steal the guitar or not? Remember, you're trying to find the set of items to steal that will give you the most value.

Esta es la fila de la guitarra, lo que significa que estás intentando meter la guitarra en la mochila. En cada celda hay una decisión simple: ¿robas la guitarra o no? Recuerde, está tratando de encontrar el conjunto de elementos para robar que le brindarán el mayor valor.

The first cell has a knapsack of capacity 1 lb. The guitar is also 1 lb, which means it fits into the knapsack! So the value of this cell is \$1,500, and it contains a guitar.

La primera celda tiene una mochila con capacidad de 1 libra. La guitarra también pesa 1 libra, lo que significa que cabe en la mochila. Entonces, el valor de esta celda es \$1,500 y contiene una guitarra.

Let's start filling in the grid.

Comencemos a completar la cuadrícula.

	1	2	3	4
GUITAR	\$1500 G			
STEREO				
LAPTOP				

Like this, each cell in the grid will contain a list of all the items that fit into the knapsack at that point.

De esta manera, cada celda de la cuadrícula contendrá una lista de todos los elementos que caben en la mochila en ese punto.

Let's look at the next cell. Here you have a knapsack with a capacity of 2 lb. Well, the guitar will definitely fit in there!

Miremos la siguiente celda. Aquí tienes una mochila con una capacidad de 2 libras. Bueno, la guitarra definitivamente cabe allí!

	1	2	3	4
GUITAR	\$1500 G	\$1500 G		
STEREO				
LAPTOP				

The same is true for the rest of the cells in this row. Remember, this is the first row, so you have *only* the guitar to choose from. You're pretending that the other two items aren't available to steal right now.

Lo mismo ocurre con el resto de las celdas de esta fila. Recuerda, esta es la primera fila, por lo que solo tienes la guitarra para elegir. Estás fingiendo que los otros dos elementos no están disponibles para robar en este momento.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO				
LAPTOP				

At this point, you're probably confused. *Why* are you doing this for knapsacks with a capacity of 1 lb, 2 lb, and so on, when the problem talks about a 4 lb knapsack? Remember how I told you that dynamic programming starts with a small problem and builds up to the big problem? You're solving subproblems here that will help you to solve the big problem. Read on, and things will become clearer.

En este punto, probablemente esté confundido. ¿Por qué haces esto con mochilas con una capacidad de 1 lb, 2 lb, etc., cuando el problema habla de una mochila de 4 lb? ¿Recuerdas que te dije que la programación dinámica comienza con un pequeño problema y llega hasta el gran problema? Aquí estás resolviendo subproblemas que te ayudarán a resolver el gran problema. Continúe leyendo y las cosas quedarán más claras.

At this point, your grid should look like this.

En este punto, su cuadricula debería verse así.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO				
LAPTOP				

Remember, you're trying to maximize the value of the knapsack. *This row represents the current best guess for this max.* So right now, according to this row, if you had a knapsack of capacity 4 lb, the max value you could put in there would be \$1,500.

Recuerde, está intentando maximizar el valor de la mochila. Esta fila representa la mejor estimación actual para este máximo. Entonces, en este momento, de acuerdo con esta fila, si tuvieras una mochila con capacidad de 4 libras, el valor máximo que podrías poner allí sería \$1,500.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO				
LAPTOP				

← OUR CURRENT  
BEST GUESS  
FOR WHAT THE  
THIEF SHOULD STEAL:  
THE GUITAR  
FOR \$1500

You know that's not the final solution. As we go through the algorithm, you'll refine your estimate.

Sabes que esa no es la solución final. A medida que avancemos en el algoritmo, perfeccionará su estimación.

## THE STEREO ROW

### LA FILA ESTÉREO

Let's do the next row. This one is for the stereo. Now that you're on the second row, you can steal the stereo or the guitar. At every row, you can steal the item at that row or the items in the rows above it. So you can't choose to steal the laptop right now, but you can steal the stereo and/or the guitar. Let's start with the first cell, a knapsack of capacity 1 lb. The current max value you can fit into a knapsack of 1 lb is \$1,500.

Hagamos la siguiente fila. Este es para el estéreo. Ahora que estás en la segunda fila, puedes robar el estéreo o la guitarra. En cada fila, puedes robar el artículo de esa fila o los artículos de las filas superiores. Así que no puedes elegir robar la computadora portátil ahora mismo, pero puedes robar el estéreo y/o la guitarra. Comencemos con la primera celda, una mochila con capacidad de 1 libra. El valor máximo actual que puede caber en una mochila de 1 libra es \$1,500.

CURRENT MAX FOR A 1lb KNAPSACK

GUITAR

STEREO

LAPTOP

NEW MAX FOR A 1lb KNAPSACK

1	2	3	4
\$1500 G	\$1500 G	\$1500 G	\$1500 G

Should you steal the stereo or not?

¿Deberías robar el estéreo o no?

You have a knapsack of capacity 1 lb. Will the stereo fit in there? Nope, it's too heavy! Because you can't fit the stereo, \$1,500 *remains* the max guess for a 1 lb knapsack.

Tienes una mochila con capacidad de 1 libra. ¿Cabe el estéreo allí? ¡No, es demasiado pesado! Debido a que no cabe el estéreo, \$1,500 sigue siendo la estimación máxima para una mochila de 1 libra.

	1	2	3	4
GUITAR	\$1500 ↓ G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G			
LAPTOP				

The same thing is true for the next two cells. These knapsacks have a capacity of 2 lb and 3 lb. The old max value for both was \$1,500.

Lo mismo ocurre con las dos celdas siguientes. Estas mochilas tienen una capacidad de 2 libras y 3 libras. El valor máximo anterior para ambas era de \$1,500.

	1	2	3	4
GUITAR	\$1500 ↓ G	\$1500 ↓ G	\$1500 ↓ G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	
LAPTOP				

The stereo still doesn't fit, so your guesses remain unchanged.

El estéreo todavía no encaja, por lo que tus conjeturas no cambian.

What if you have a knapsack of capacity 4 lb? Aha! The stereo finally fits! The old max value was \$1,500, but if you put the stereo in there instead, the value is \$3,000! Let's take the stereo.

¿Qué pasa si tienes una mochila con capacidad de 4 libras? ¡Ajá! ¡El estéreo finalmente encaja! El antiguo valor máximo era \$1,500, pero si colocas el estéreo allí, el valor es \$3,000! Tomemos el estéreo.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 G '''5
LAPTOP				

You just updated your estimate! If you have a 4 lb knapsack, you can fit at least \$3,000 worth of goods in it. You can see from the grid that you're incrementally updating your estimate.

¡Acabas de actualizar tu estimación! Si tiene una mochila de 4 libras, puede guardar mercancías por un valor de al menos \$ 3,000 en ella. Puede ver en la cuadrícula que está actualizando gradualmente su estimación.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 115
LAPTOP				

← OLD ESTIMATE  
← NEW ESTIMATE  
← FINAL SOLUTION

## THE LAPTOP ROW

### LA FILA DE PORTÁTILES

Let's do the same thing with the laptop! The laptop weighs 3 lb, so it won't fit into a 1 lb or a 2 lb knapsack. The estimate for the first two cells stays at \$1,500.

Hagamos lo mismo con la computadora portátil! La computadora portátil pesa 3 libras, por lo que no cabe en una mochila de 1 libra o 2 libras. La estimación para las dos primeras celdas se mantiene en 1.500 dólares.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 S
LAPTOP	\$1500 G	\$1500 G		

At 3 lb, the old estimate was \$1,500. But you can choose the laptop instead, and that's worth \$2,000. So the new max estimate is \$2,000!

Con 3 libras, la estimación anterior era de 1.500 dólares. Pero puedes elegir la computadora portátil en su lugar, y eso vale \$2,000. ¡Así que la nueva estimación máxima es \$2,000!

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 S
LAPTOP	\$1500 G	\$1500 G	\$2000 L	

At 4 lb, things get really interesting. This is an important part. The current estimate is \$3,000. You can put the laptop in the knapsack, but it's only worth \$2,000.

Con 4 libras, las cosas se ponen realmente interesantes. Esta es una parte importante. La estimación actual es de 3.000 dólares. Puedes poner el portátil en la mochila, pero sólo vale 2.000 dólares.

**\$3000** vs **\$2000**  
STEREO LAPTOP

Hmm, that's not as good as the old estimate. But wait! The laptop weighs only 3 lb, so you have 1 lb free! You could put something in this 1 lb.

Mmmm, eso no es tan bueno como la estimación anterior. ¡Pero espera! La computadora portátil pesa solo 3 libras, ¡así que tienes 1 libra gratis! Podrías poner algo en este 1 lb.

**\$3000** vs **( \$2000 + ??? )**  
STEREO LAPTOP  
1 LB OF FREE SPACE

What's the maximum value you can fit into 1 lb of space? Well, you've been calculating it all along.

¿Cuál es el valor máximo que puede caber en 1 libra de espacio? Bueno, lo has estado calculando todo el tiempo.

1    2    3    4

	1	2	3	4	
	\$1500	\$1500	\$1500	\$1500	
	G	G	G	G	
MAX VALUE FOR 1lb	→	\$1500	\$1500	\$1500	\$3000
	G	G	G	S	
	↓	↓	↓		
	\$1500	\$1500	\$1500	\$2000	
	G	G	G	L	

According to the last best estimate, you can fit the guitar into that 1 lb space, and that's worth \$1,500. So the real comparison is as follows.

Según la última mejor estimación, puedes colocar la guitarra en ese espacio de 1 libra, y eso vale \$1,500. Entonces la comparación real es la siguiente.

$$\begin{matrix} \$3000 & \text{vs} & (\$2000 + \$1500) \\ \text{STEREO} & & \text{LAPTOP} & \text{GUITAR} \end{matrix}$$

You might have been wondering why you were calculating max values for smaller knapsacks. I hope now it makes sense! When you have space left over, you can use the answers to those subproblems to figure out what will fit in that space. It's better to take the laptop + the guitar for \$3,500.

Quizás te hayas preguntado por qué estabas calculando valores máximos para mochilas más pequeñas. ¡Espero que

ahora tenga sentido! Cuando te quede espacio, puedes usar las respuestas a esos subproblemas para descubrir qué cabe en ese espacio. Es mejor llevarse el portátil + la guitarra por 3.500 dólares.

The final grid looks like this.

La cuadrícula final se ve así.

	1	2	3	4
GUITAR	\$1500 ↓ G	\$1500 ↓ G	\$1500 ↓ G	\$1500 G
STEREO	\$1500	\$1500	\$1500 G	\$3000
LAPTOP	\$1500 ↓ G	\$1500 ↓ G	\$2000 L	\$3500 L G
				↑ THE ANSWER!

There's the answer: the maximum value that will fit in the knapsack is \$3,500, made up of a guitar and a laptop!

Aquí está la respuesta: el valor máximo que cabe en la mochila es de 3.500 dólares, icompuesta por una guitarra y un portátil!

Maybe you think I used a different formula to calculate the value of that last cell. That's because I skipped some unnecessary complexity when filling in the values of the earlier cells. Each cell's value gets calculated with the same formula. Here it is.

Quizás pienses que usé una fórmula diferente para calcular el valor de esa última celda. Esto se debe a que me salté algunas complejidades innecesarias al completar los valores de las celdas anteriores. El valor de cada celda se calcula con la misma fórmula. Aquí lo tienes.

$$\text{CELL}[i][j] = \max \text{ of } \left\{ \begin{array}{l} 1. \text{ THE PREVIOUS MAX (VALUE AT CELL } [i-1][j] \text{)} \\ \quad \quad \quad \text{vs} \\ 2. \text{ VALUE OF CURRENT ITEM + VALUE OF THE REMAINING SPACE} \\ \quad \quad \quad \uparrow \\ \quad \quad \quad \text{CELL}[i-1][j - \text{ITEM'S WEIGHT}] \end{array} \right\}$$

ROW      COLUMN  
↓          ↓

You can use this formula with every cell in this grid, and you should end up with the same grid I did. Remember how I talked about solving subproblems? You combined the solutions to two subproblems to solve the bigger problem.

Puedes usar esta fórmula con cada celda de esta cuadrícula y deberías terminar con la misma cuadrícula que yo hice. ¿Recuerdas que hablé de resolver subproblemas? Combinaste las soluciones de dos subproblemas para resolver el problema más grande.



## ***Knapsack problem FAQ***

### ***Preguntas frecuentes sobre problemas de mochila***

Maybe this still feels like magic. This section answers some common questions.

Quizás esto todavía parezca magia. Esta sección responde algunas preguntas comunes.

#### **What happens if you add an item?**

#### **¿Qué pasa si agregas un artículo?**

Suppose you realize there's a fourth item you can steal that you didn't notice before! You can also steal an iPhone.

iSupongamos que te das cuenta de que hay un cuarto objeto que puedes robar y que no habías notado antes! También

puedes robar un iPhone.



IPHONE  
\$2000  
1lb

Do you have to recalculate everything to account for this new item? Nope. Remember, dynamic programming keeps progressively building on your estimate. So far, these are the max values.

¿Tiene que volver a calcular todo para tener en cuenta este nuevo elemento? No. Recuerde, la programación dinámica sigue basándose progresivamente en su estimación. Hasta ahora, estos son los valores máximos.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 S
LAPTOP	\$1500 G	\$1500 G	\$2000 L	\$3500 LG

That means for a 4 lb knapsack, you can steal \$3,500 worth of goods. You thought that was the final max value. But let's add a row for the iPhone.

Eso significa que por una mochila de 4 libras, puedes robar bienes por valor de \$3,500. Pensaste que ese era el valor máximo final. Pero agreguemos una fila para el iPhone.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 S
LAPTOP	\$1500 G	\$1500 G	\$2000 L	\$3500 LG
IPHONE				

↑  
NEW ANSWER

Turns out you have a new max value! Try to fill in this new row before reading on.

¡Resulta que tienes un nuevo valor máximo! Intenta completar esta nueva fila antes de seguir leyendo.

Let's start with the first cell. The iPhone fits into the 1 lb knapsack. The old max was \$1,500, but the iPhone is worth \$2,000. Let's take the iPhone instead.

Comencemos con la primera celda. El iPhone cabe en la mochila de 1 libra. El antiguo máximo era de 1.500 dólares, pero el iPhone vale 2.000 dólares. En su lugar, tomemos el iPhone.

	1	2	3	4
GUITAR	\$1500 G	\$1500 G	\$1500 G	\$1500 G
STEREO	\$1500 G	\$1500 G	\$1500 G	\$3000 S
LAPTOP	\$1500 G	\$1500 G	\$2000 L	\$3500 LG
IPHONE	\$2000 I			

In the next cell, you can fit the iPhone *and* the guitar.

En la siguiente celda cabe el iPhone y la guitarra.

\$1500 G	\$1500 G	\$1500 G	\$1500 G
\$1500 G	\$1500 G	\$1500 G	\$3000 S
\$1500 G	\$1500 G	\$2000 L	\$3500 LG
\$2000 I	\$3500 IG		

For cell 3, you can't do better than take the iPhone and the guitar again, so leave it as is.

Para el celular 3, no puedes hacer nada mejor que volver a coger el iPhone y la guitarra, así que déjalo como está.

For the last cell, things get interesting. The current max is \$3,500. You can steal the iPhone instead, and you have 3 lb of space left over.

Para la última celda, las cosas se ponen interesantes. El máximo actual es de \$3,500. En su lugar, puedes robar el iPhone y te sobrarán 3 libras de espacio.

$$\begin{array}{c} \$3500 \\ \text{LAPTOP + GUITAR} \end{array} \quad \text{vs} \left( \begin{array}{c} \$2000 \\ \text{IPHONE} \end{array} + \underbrace{\begin{array}{c} ? ? ? \\ \text{3 LBS FREE} \end{array}} \right)$$

Those 3 lb are worth \$2,000! \$2,000 from the iPhone + \$2,000 from the old subproblem: that's \$4,000. A new max!

¡Esas 3 libras valen \$2,000! 2.000 dólares del iPhone + 2.000 dólares del antiguo subproblema: son 4.000 dólares.  
¡Un nuevo máximo!

Here's the new final grid.

Aquí está la nueva cuadrícula final.

\$1500 G	\$1500 G	\$1500 G	\$1500 G
\$1500 G	\$1500 G	\$1500 G	\$3000 S
\$1500 G	\$1500 G	\$2000 L	\$3500 LG
\$2000 I	\$3500 IG	\$3500 IG	\$4000 IL

↑  
NEW  
ANSWER

Question: Would the value of a column ever go *down*? Is this possible?

Pregunta: ¿Bajaría alguna vez el valor de una columna? es posible?

1      2      3      4

MAX VALUE DECREASED AS WE WENT ON

\$1500	\$1500	\$1500	\$1500
∅	∅	∅	\$3000

Think of an answer before reading on.

Piensa en una respuesta antes de seguir leyendo.

Answer: No. At every iteration, you're storing the current max estimate. The estimate can never get worse than it was before!

Respuesta: No. En cada iteración, se almacena la estimación máxima actual. La estimación nunca podrá ser peor que antes!

## EXERCISE

### EJERCICIO

**11.1** Suppose you can steal another item: a mechanical keyboard player. It weighs 1 lb and is worth \$1,000. Should you steal it?

11.1 Supongamos que puedes robar otro objeto: un tecladista mecánico. Pesa 1 libra y vale \$1,000. ¿Deberías robarlo?

## What happens if you change the order of the rows?

### ¿Qué pasa si cambias el orden de las filas?

Does the answer change? Suppose you fill the rows in this order: stereo, laptop, guitar. What does the grid look like? Fill out the grid for yourself before moving on.

¿Cambia la respuesta? Suponga que completa las filas en este orden: estéreo, computadora portátil, guitarra. ¿Cómo

se ve la cuadrícula? Complete la cuadrícula usted mismo antes de continuar.

Here's what the grid looks like.

Así es como se ve la cuadrícula.

	1	2	3	4
STEREO	Ø	Ø	Ø	\$3000 S
LAPTOP	Ø	Ø	\$2000 L	\$3000 S
GUITAR	\$1500 G	\$1500 G	\$2000 L	\$3500 LG

The answer doesn't change. The order of the rows doesn't matter.

La respuesta no cambia. El orden de las filas no importa.

**Can you fill in the grid column-wise instead of row-wise?**

**¿Puedes completar la cuadrícula por columnas en lugar de por filas?**

Try it for yourself! For this problem, it doesn't make a difference. It could make a difference for other problems.

¡Pruébalo tú mismo! Para este problema, no hay diferencia. Podría marcar la diferencia para otros problemas.

## What happens if you add a smaller item?

### ¿Qué pasa si agregas un elemento más pequeño?

Suppose you can steal a necklace. It weighs 0.5 lb, and it's worth \$1,000. So far, your grid assumes that all weights are integers. Now you decide to steal the necklace. You have 3.5 lb left over. What's the max value you can fit in 3.5 lb? You don't know! You only calculated values for 1 lb, 2 lb, 3 lb, and 4 lb knapsacks. You need to know the value of a 3.5 lb knapsack.

Supongamos que puedes robar un collar. Pesa 0,5 libras y vale 1.000 dólares. Hasta ahora, su cuadrícula supone que todos los pesos son números enteros. Ahora decides robar el collar. Te sobran 3,5 libras. ¿Cuál es el valor máximo que cabe en 3,5 libras? ¡No lo sabes! Solo calculó valores para mochilas de 1 lb, 2 lb, 3 lb y 4 lb. Necesitas saber el valor de una mochila de 3,5 libras.

*Because of the necklace, you have to account for finer granularity, so the grid has to change.*

Debido al collar, hay que tener en cuenta una granularidad más fina, por lo que la cuadrícula tiene que cambiar.

	0.5	1	1.5	2	2.5	3	3.5	4
GUITAR								
STEREO								
LAPTOP								
JEWELRY								

## Can you steal fractions of an item?

### ¿Puedes robar fracciones de un artículo?

Suppose you're a thief in a grocery store. You can steal bags of lentils and rice. If a whole bag doesn't fit, you can open it and take as much as you can carry. So now it's not all or nothing—you can take a fraction of an item. How do you handle this using dynamic programming?

Supongamos que eres un ladrón en una tienda de comestibles. Puedes robar bolsas de lentejas y arroz. Si no cabe un bolso entero, puedes abrirlo y coger todo lo que puedas llevar. Así que ahora no es todo o nada: puedes tomar una fracción de un artículo. ¿Cómo se maneja esto usando programación dinámica?

Answer: You can't. With the dynamic programming solution, you either take the item or not. There's no way for it to figure out that you should take half an item.

Respuesta: No puedes. Con la solución de programación dinámica, tomas el artículo o no. No hay forma de que se dé cuenta de que debes llevarte la mitad de un artículo.

But this case is easily solved using a greedy algorithm! First, take as much as you can of the most valuable item. When that runs out, take as much as you can of the next most valuable item, and so on.

¡Pero este caso se resuelve fácilmente usando un algoritmo codicioso! Primero, toma todo lo que puedas del artículo más valioso. Cuando se acabe, toma todo lo que puedas del siguiente artículo más valioso, y así sucesivamente.

For example, suppose you have these items to choose from.

Por ejemplo, supongamos que tiene estos elementos para elegir.



Quinoa is more expensive per pound than anything else. So, take all the quinoa you can carry! If that fills your knapsack, that's the best you can do.

La quinua es más cara por libra que cualquier otra cosa. Así que lleva toda la quinua que puedas llevar! Si eso llena tu mochila, es lo mejor que puedes hacer.



If the quinoa runs out and you still have space in your knapsack, take the next most valuable item, and so on.

Si se acaba la quinua y todavía tienes espacio en tu mochila, toma el siguiente artículo más valioso, y así sucesivamente.

## Optimizing your travel itinerary

### Optimización de su itinerario de viaje

Suppose you're going to London for a nice vacation. You have two days there and a lot of things you want to do. You can't do everything, so you make a list.

Suponga que va a Londres para pasar unas agradables vacaciones. Tienes dos días allí y muchas cosas que quieres hacer. No puedes hacerlo todo, entonces haces una lista.

ATTRACTION	TIME	RATING
WESTMINSTER ABBEY	1/2 DAY	7
GLOBE THEATER	1/2 DAY	6
NATIONAL GALLERY	1 DAY	9
BRITISH MUSEUM	2 DAYS	9
ST. PAUL'S CATHEDRAL	1/2 DAY	8

For each thing you want to see, you write down how long it will take and rate how much you want to see it. Can you figure out what you should visit based on this list?

Para cada cosa que quieras ver, anotas cuánto tiempo te llevará y puntúas cuántoquieres verlo. ¿Puedes averiguar qué deberías visitar basándote en esta lista?

It's the knapsack problem again! Instead of a knapsack, you have a limited amount of time. And instead of stereos and laptops, you have a list of places you want to go. Draw the dynamic programming grid for this list before moving on.

¡Es el problema de la mochila otra vez! En lugar de una mochila, tienes un tiempo limitado. Y en lugar de estéreos y portátiles, tienes una lista de lugares a los que quieres ir. Dibuja la cuadrícula de programación dinámica para esta lista antes de continuar.

Here's what the grid looks like.

Así es como se ve la cuadrícula.

$\frac{1}{2}$	1	$1\frac{1}{2}$	2
WESTMINSTER			
GLOBE THEATER			
NATIONAL GALLERY			
BRITISH MUSEUM			
ST. PAUL'S			

Did you get it right? Fill in the grid. What places should you end up visiting? Here's the answer.

¿Lo entendiste correctamente? Rellena en la cuadrícula. ¿Qué lugares deberías terminar visitando? Aquí está la respuesta.

	$\frac{1}{2}$	1	$1\frac{1}{2}$	2
WESTMINSTER	7w	7w	7w	7w
GLOBE THEATER	7w	13WG	13WG	13WG
NATIONAL GALLERY	7w	13WG	16WN	22WGN
BRITISH MUSEUM	7w	13WG	16WN	22WN
ST PAUL'S	8s	15WS	21WGS	24WNS

↑  
FINAL ANSWER:  
WESTMINSTER ABBEY,  
NATIONAL GALLERY,  
ST. PAUL'S CATHEDRAL

## Handling items that depend on each other

**Manejo de elementos que dependen unos de otros.**

Suppose you want to go to Paris, so you add a couple of items to the list.

Supongamos que quieres ir a París, entonces agregas un par de elementos a la lista.

EIFFEL TOWER	$\frac{1}{2}$ DAY	8
THE LOUVRE	$\frac{1}{2}$ DAY	9
NOTRE DAME	$\frac{1}{2}$ DAY	7

These places take a lot of time because first you have to travel from London to Paris. That takes half a day. If you want to do all three items, it will take 4.5 days.

Estos lugares toman mucho tiempo porque primero hay que viajar de Londres a París. Eso lleva medio día. Si desea realizar los tres elementos, le llevará 4,5 días.

Wait, that's not right. You don't have to travel to Paris for each item. Once you're in Paris, each item should only take a day. So it should be one day per item + half a day of travel = 3.5 days, not 4.5 days.

Espera, eso no está bien. No es necesario viajar a París para cada artículo. Una vez que estés en París, cada artículo solo debería llevarte un día. Por lo tanto, debería ser un día por artículo + medio día de viaje = 3,5 días, no 4,5 días.

If you put the Eiffel Tower in your knapsack, the Louvre becomes "cheaper"—it will only cost you a day instead of 1.5 days. How do you model this in dynamic programming?

Si pones la Torre Eiffel en tu mochila, el Louvre se vuelve "más barato": sólo te costará un día en lugar de 1,5 días. ¿Cómo se modela esto en programación dinámica?

You can't. Dynamic programming is powerful because it can solve subproblems and use those answers to solve the big

problem. *Dynamic programming only works when each subproblem is discrete—when it doesn't depend on other subproblems.* That means there's no way to account for París using the dynamic programming algorithm.

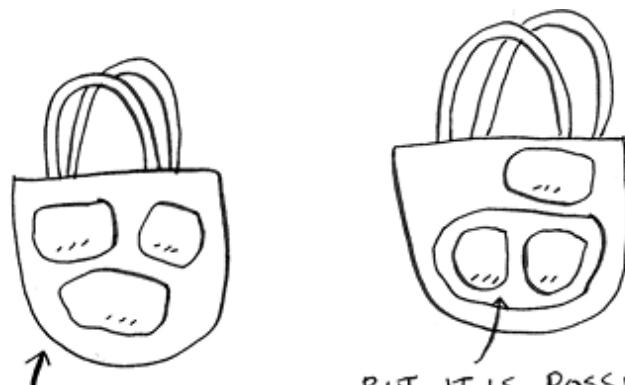
No puedes. La programación dinámica es poderosa porque puede resolver subproblemas y utilizar esas respuestas para resolver el gran problema. La programación dinámica sólo funciona cuando cada subproblema es discreto, es decir, cuando no depende de otros subproblemas. Eso significa que no hay forma de contabilizar París utilizando el algoritmo de programación dinámica.

## **Is it possible that the solution will require more than two sub-knapsacks?**

## **¿Es posible que la solución requiera más de dos submochilas?**

It's possible that the best solution involves stealing more than two items. The way the algorithm is set up, you're combining two knapsacks at most—you'll never have more than two sub-knapsacks. But it's possible for those sub-knapsacks to have their own sub-knapsacks.

Es posible que la mejor solución implique robar más de dos elementos. Según la forma en que está configurado el algoritmo, estás combinando dos mochilas como máximo; nunca tendrás más de dos submochilas. Pero es posible que esas submochilas tengan sus propias submochilas.



IT'S NOT POSSIBLE  
TO HAVE THREE  
SUB-KNAPSACKS

BUT IT IS POSSIBLE  
TO HAVE SUB-KNAPSACKS  
THAT HAVE THEIR OWN  
SUB-KNAPSACKS

**Is it possible that the best solution doesn't fill the knapsack completely?**

**¿Es posible que la mejor solución no llene la mochila del todo?**

Yes. Suppose you could also steal a diamond.

Sí. Supongamos que también pudieras robar un diamante.



This is a big diamond: it weighs 3.5 pounds. It's worth a million dollars, way more than anything else. You should definitely steal it! But there's half a pound of space left, and nothing will fit in that space.

Este es un diamante grande: pesa 3,5 libras. Vale un millón de dólares, mucho más que cualquier otra cosa.  
¡Definitivamente deberías robarlo! Pero queda media libra de espacio y nada cabe en ese espacio.

## EXERCISE

## EJERCICIO

**11.2** Suppose you're going camping. You have a knapsack that will hold 6 lb, and you can take the following items. Each has a value, and the higher the value, the more important the item is:

11.2 Supongamos que vas a acampar. Tienes una mochila con capacidad para 6 libras y puedes llevar los siguientes artículos. Cada uno tiene un valor y

cuanto mayor sea el valor, más importante será el artículo:

- Water, 3 lb, 10  
Agua, 3 libras, 10
- Book, 1 lb, 3  
Libro, 1 libra, 3
- Food, 2 lb, 9  
Comida, 2 libras, 9
- Jacket, 2 lb, 5  
Chaqueta, 2 libras, 5
- Camera, 1 lb, 6  
Cámara, 1 libra, 6

What's the optimal set of items to take on your camping trip?

¿Cuál es el conjunto óptimo de artículos para llevar en su viaje de campamento?

## ***Longest common substring***

## ***Subcadena común más larga***

You've seen one dynamic programming problem so far. What are the takeaways?

Hasta ahora has visto un problema de programación dinámica. ¿Cuáles son las conclusiones?



- Dynamic programming is useful *when you're trying to optimize something given a constraint*. In the knapsack problem, you had to maximize the value of the goods you stole, constrained by the size of the knapsack.  
La programación dinámica es útil cuando intentas optimizar algo dada una restricción. En el problema de la mochila, había que maximizar el valor de los bienes que robaba, limitado por el tamaño de la mochila.
- You can use dynamic programming when the problem can be broken into discrete subproblems, and they don't depend on each other.  
Puede utilizar la programación dinámica cuando el problema se puede dividir en subproblemas discretos y no dependen unos de otros.

It can be hard to come up with a dynamic programming solution. That's what we'll focus on in this section. Some general tips follow:

Puede resultar difícil encontrar una solución de programación dinámica. En eso nos centraremos en esta sección. A continuación se presentan algunos consejos generales:

- It's often useful to picture a dynamic programming problem as a grid.

A menudo resulta útil imaginar un problema de programación dinámica como una cuadrícula.
- The values in the cells are usually what you're trying to optimize. For the knapsack problem, the values are the value of the goods.

Los valores de las celdas suelen ser los que intentas optimizar. Para el problema de la mochila, los valores son el valor de las mercancías.
- Each cell is a subproblem, so think about how you can divide your problem into subproblems. That will help you figure out what the axes are.

Cada celda es un subproblema, así que piensa en cómo puedes dividir tu problema en subproblemas. Eso te ayudará a descubrir cuáles son los ejes.

Let's look at another example. Suppose you're running dictionary.com. Someone types in a word, and you give them the definition.

Veamos otro ejemplo. Supongamos que está ejecutando diccionario.com. Alguien escribe una palabra y usted le da la definición.



But if someone misspells a word, you want to be able to guess what word they meant. Alex is searching for *fish*, but he accidentally put in *hish*. That's not a word in your dictionary, but you have a list of words that are similar.

Pero si alguien escribe mal una palabra, querrás poder adivinar qué palabra quiso decir. Alex está buscando pescado, pero accidentalmente lo puso en él. Esa no es una palabra en su diccionario, pero tiene una lista de palabras que son similares.

### SIMILAR TO "HISH":

- FISH
- VISTA

(This is a toy example, so you'll limit your list to two words. In reality, this list would probably be thousands of words.)

(Este es un ejemplo de juguete, por lo que limitará su lista a dos palabras. En realidad, esta lista probablemente tendría miles de palabras).

Alex typed *hish*. Which word did Alex mean to type: *fish* or *vista*?

Alex escribió *hish*. ¿Qué palabra quiso escribir Alex: *pez* o *vista*?

## Making the grid

### haciendo la grilla

What does the grid for this problem look like? You need to answer these questions:

¿Cómo se ve la cuadrícula para este problema? Necesitas responder estas preguntas:

- What are the values of the cells?  
¿Cuáles son los valores de las celdas?
- How do you divide this problem into subproblems?  
¿Cómo se divide este problema en subproblemas?
- What are the axes of the grid?  
¿Cuáles son los ejes de la cuadrícula?

In dynamic programming, you're trying to *maximize* something. In this case, you're trying to find the longest substring that two words have in common. What substring do *hish* and *fish* have in common? How about *hish* and *vista*? That's what you want to calculate.

En la programación dinámica, estás intentando maximizar algo. En este caso, estás intentando encontrar la subcadena

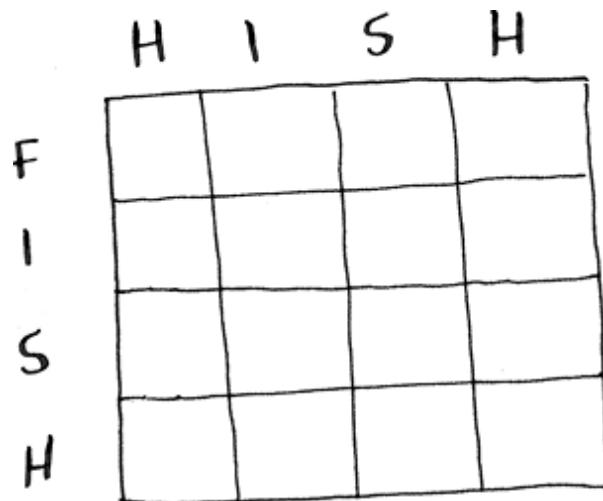
más larga que dos palabras tienen en común. ¿Qué subcadena tienen en común *hish* y *fish*? ¿Qué tal *hish* y *vista*? Eso es lo que quieras calcular.

Remember, the values for the cells are usually what you're trying to optimize. In this case, the values will probably be a number: the length of the longest substring that the two strings have in common.

Recuerde, los valores de las celdas suelen ser los que intenta optimizar. En este caso, los valores probablemente serán un número: la longitud de la subcadena más larga que las dos cadenas tienen en común.

How do you divide this problem into subproblems? You could compare substrings. Instead of comparing *hish* and *fish*, you could compare *his* and *fis* first. Each cell will contain the length of the longest substring that two substrings have in common. This also gives you a clue that the axes will probably be the two words. So the grid probably looks like this.

¿Cómo se divide este problema en subproblemas? Podrías comparar subcadenas. En lugar de comparar *hish* y *fish*, puedes comparar primero *hish* y *fis*. Cada celda contendrá la longitud de la subcadena más larga que dos subcadenas tengan en común. Esto también te da una pista de que los ejes probablemente serán las dos palabras. Entonces la cuadrícula probablemente se vea así.



If this seems like black magic to you, don't worry. This is hard stuff—that's why I'm teaching it so late in the book! Later, I'll give you an exercise so you can practice dynamic programming yourself.

Si esto te parece magia negra, no te preocupes. Esto es algo difícil, ipor eso lo enseño tan tarde en el libro! Más adelante te daré un ejercicio para que puedas practicar tú mismo la programación dinámica.

## Filling in the grid

## Llenando la grilla

Now you have a good idea of what the grid should look like. What's the formula for filling in each cell of the grid? You can cheat a little because you already know what the solution should be—*hish* and *fish* have a substring of length 3 in common: *ish*.

Ahora tienes una buena idea de cómo debería verse la cuadrícula. ¿Cuál es la fórmula para completar cada celda de la cuadrícula? Puedes hacer un poco de trampa porque ya sabes cuál debería ser la solución: hish y fish tienen una subcadena de longitud 3 en común: ish.

But that still doesn't tell you the formula to use. Computer scientists sometimes joke about using the Feynman algorithm. The *Feynman algorithm* is named after the famous physicist Richard Feynman, and it works like this:

Pero eso todavía no te dice cuál es la fórmula a utilizar. Los informáticos a veces bromean sobre el uso del algoritmo de Feynman. El algoritmo de Feynman lleva el nombre del famoso físico Richard Feynman y funciona así:

1. Write down the problem.

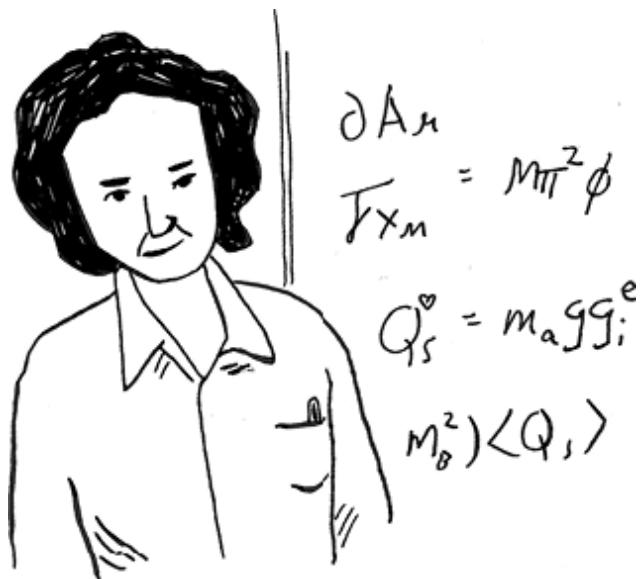
Escribe el problema.

2. Think real hard.

Piensa muy bien.

3. Write down the solution.

Escribe la solución.



Computer scientists are a fun bunch!

¡Los informáticos son un grupo divertido!

The truth is that there's no easy way to calculate the formula here. You have to experiment and try to find something that works. Sometimes algorithms aren't an exact recipe. They're a framework that you build your idea on top of.

La verdad es que no existe una manera fácil de calcular la fórmula aquí. Tienes que experimentar e intentar encontrar algo que funcione. A veces los algoritmos no son una receta exacta. Son un marco sobre el que construyes tu idea.

Try to come up with a solution to this problem yourself. I'll give you a hint—part of the grid looks like this.

Intente encontrar usted mismo una solución a este problema. Te daré una pista: parte de la cuadricula se ve así.

	H	I	S	H
F	0	0		
I				
S			2	0
H				3

What are the other values? Remember that each cell is the value of a *subproblem*. Why does cell (3, 3) have a value of 2? Why does cell (3, 4) have a value of 0?

¿Cuáles son los otros valores? Recuerda que cada celda es el valor de un subproblema. ¿Por qué la celda (3, 3) tiene un valor de 2? ¿Por qué la celda (3, 4) tiene un valor de 0?

Read on after you've tried to come up with a formula yourself. Even if you don't get it right, my explanation will make a lot more sense.

Continúe leyendo después de haber intentado encontrar una fórmula usted mismo. Incluso si no lo haces bien, mi explicación tendrá mucho más sentido.

## The solution

### La solución

Here's the final grid.

Aquí está la cuadrícula final.

	H	I	S	H
F	0	0	0	0
I	0	1	0	0
S	0	0	2	0
H	0	0	0	3

Here's my formula for filling in each cell.

Aquí está mi fórmula para completar cada celda.

1. IF THE LETTERS  
DONT MATCH,  
THE VALUE IS  
ZERO

	H	I	S	H
F	0	0	0	0
I	0	1	0	0
S	0	0	2	0
H	1	0	0	3

2. IF THEY DO MATCH,  
THIS VALUE IS  
VALUE OF TOP-LEFT NEIGHBOR + 1

Here's how the formula looks in pseudocode:

Así es como se ve la fórmula en pseudocódigo:

```
if word_a[i] == word_b[j]:          ①
    cell[i][j] = cell[i-1][j-1] + 1
else:                                ②
    cell[i][j] = 0
```

- ① The letters match.
- ① Las letras coinciden.
- ② The letters don't match.
- ② Las letras no coinciden.

Here's the grid for *hish* vs. *vista*.

Aquí está la cuadrícula para *hish* vs. *vista*.

	V	I	S	T	A
H	0	0	0	0	0
I	0	1	0	0	0
S	0	0	2	0	0
H	0	0	0	0	0

↓                      ↑

FINAL ANSWER      NOT THE FINAL ANSWER

One thing to note: for this problem, the final solution may not be in the last cell! For the knapsack problem, this last cell always had the final solution. But for the longest common substring, the solution is the largest number in the grid—and it may not be the last cell.

Una cosa a tener en cuenta: para este problema, es posible que la solución final no esté en la última celda. Para el problema de la mochila, esta última celda siempre tenía la solución final. Pero para la subcadena común más larga, la solución es el número más grande de la cuadrícula y puede que no sea la última celda.

Let's go back to the original question: Which string has more in common with *hish*? *hish* and *fish* have a substring of three letters in common. *hish* and *vista* have a substring of two letters in common.

Volvamos a la pregunta original: ¿Qué cadena tiene más en común con *hish*? *hish* y *fish* tienen una subcadena de tres letras en común. *hish* y *vista* tienen una subcadena de dos letras en común.

Alex probably meant to type *fish*.

Alex probablemente quiso escribir pescado.

## Longest common subsequence

### Subsecuencia común más larga

Suppose Alex accidentally searched for *fosh*. Which word did he mean: *fish* or *fort*?

Supongamos que Alex busca accidentalmente pescado. ¿Qué palabra quiso decir: pez o fuerte?

Let's compare them using the longest-common-substring formula.

Comparémoslos usando la fórmula de subcadena común más larga.

	F	O	S	H
F	I	O	O	O
O	O	2	O	O
R	O	O	O	O
T	O	O	O	O

vs

	F	O	S	H
F	I	O	O	O
I	O	O	O	O
S	O	O	I	O
H	O	O	O	2

They're both the same: two letters! But *fosh* is closer to *fish*.

Ambos son iguales: idos letras! Pero el *fosh* está más cerca del pescado.

$$\begin{matrix} F & O & S & H \\ \downarrow & \downarrow & \downarrow & \\ F & I & S & H \end{matrix} = 3$$

$$\begin{matrix} F & O & S & H \\ \downarrow & \downarrow & & \\ F & O & R & T \end{matrix} = 2$$

You're comparing the longest common *substring*, but you really need to compare the longest common *subsequence*: the number of letters in a sequence that the two words have in common. How do you calculate the longest common subsequence?

Estás comparando la subcadena común más larga, pero realmente necesitas comparar la subsecuencia común más larga: el número de letras en una secuencia que las dos

palabras tienen en común. ¿Cómo se calcula la subsecuencia común más larga?

Here's the partial grid for *fish* and *fosh*.

Aquí está la cuadrícula parcial para pescado y fosh.

	F	O	S	H
F	1	1		
I	1			
S		1	2	2
H				

Can you figure out the formula for this grid? The longest common subsequence is very similar to the longest common substring, and the formulas are pretty similar, too. Try to solve it yourself—I give the answer next.

¿Puedes encontrar la fórmula para esta cuadrícula? La subsecuencia común más larga es muy similar a la subcadena común más larga y las fórmulas también son bastante similares. Intente resolverlo usted mismo; a continuación le doy la respuesta.

## Longest common subsequence—solution

## Subsecuencia común más larga: solución

Here's the final grid.

Aquí está la cuadrícula final.

	F	O	S	H
F	1 → 1 → 1 → 1			
O	↓ 1	2 → 2 → 2		
R	↓ 1	2 → 2 → 2		
T	1	2 → 2 → 2		

LONGEST COMMON  
SUBSEQUENCE = 2

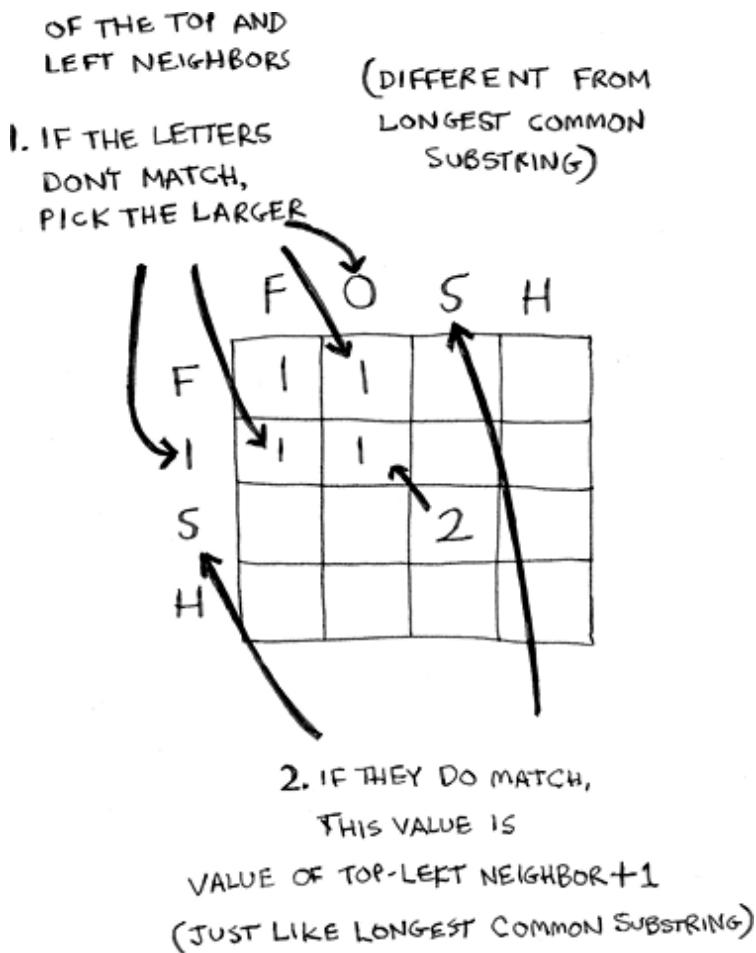
vs

	F	O	S	H
F	1 → 1 → 1 → 1			
I	↓ 1	1 → 1 → 1 → 1		
S	↓ 1	1 → 1	2 → 2	
H	↓ 1	1 → 1	2	3

LONGEST COMMON  
SUBSEQUENCE = 3

Here's my formula for filling in each cell.

Aquí está mi fórmula para completar cada celda.



And here it is in pseudocode:

Y aquí está en pseudocódigo:

```

if word_a[i] == word_b[j]:           ①
    cell[i][j] = cell[i-1][j-1] + 1
else:                                ②
    cell[i][j] = max(cell[i-1][j], cell[i][j-1])

```

- ① The letters match.
- ① Las letras coinciden.
- ② The letters don't match.
- ② Las letras no coinciden.

Whew! You did it! This is definitely one of the toughest chapters in the book. So is dynamic programming ever really used? The answer is yes:

¡Uf! ¡Lo hiciste! Este es definitivamente uno de los capítulos más difíciles del libro. Entonces, ¿alguna vez se utiliza realmente la programación dinámica? La respuesta es sí:

- Biologists use the longest common subsequence to find similarities in DNA strands. They can use this to tell how similar two animals or two diseases are. The longest common subsequence is being used to find a cure for multiple sclerosis.

Los biólogos utilizan la subsecuencia común más larga para encontrar similitudes en las cadenas de ADN.

Pueden usar esto para saber qué tan similares son dos animales o dos enfermedades. La subsecuencia común más larga se está utilizando para encontrar una cura para la esclerosis múltiple.

- Have you ever used `diff` (like `git diff`)? `Diff` tells you the differences between two files, and it uses dynamic programming to do so.

¿Alguna vez has usado `diff` (como `git diff`)? `Diff` le indica las diferencias entre dos archivos y utiliza programación dinámica para hacerlo.

- We talked about string similarity. *Levenshtein distance* measures how similar two strings are, and it uses dynamic programming. Levenshtein distance is used for everything from spell-check to figuring out whether a user is uploading copyrighted data.

Hablamos de similitud de cadenas. La distancia de Levenshtein mide qué tan similares son dos cadenas y utiliza programación dinámica. La distancia de Levenshtein se utiliza para todo, desde revisar la ortografía hasta determinar si un usuario está cargando datos protegidos por derechos de autor.

## EXERCISE

### EJERCICIO

**11.3** Draw and fill in the grid to calculate the longest common substring between *blue* and *clues*.

11.3 Dibuje y complete la cuadrícula para calcular la subcadena común más larga entre el azul y las pistas.

## **Recap**

### **Resumen**

- Dynamic programming is useful when you're trying to optimize something given a constraint.  
La programación dinámica es útil cuando intentas optimizar algo dada una restricción.
- You can use dynamic programming when the problem can be broken into discrete subproblems.  
Puede utilizar la programación dinámica cuando el problema se puede dividir en subproblemas discretos.
- Every dynamic programming solution involves a grid.  
Cada solución de programación dinámica implica una cuadrícula.
- The values in the cells are usually what you're trying to optimize.  
Los valores de las celdas suelen ser los que intentas optimizar.
- Each cell is a subproblem, so think about how you can divide your problem into subproblems.  
Cada celda es un subproblema, así que piensa en cómo puedes dividir tu problema en subproblemas.
- There's no single formula for calculating a dynamic programming solution.  
No existe una fórmula única para calcular una solución de programación dinámica.

# **12 k-nearest neighbors**

---

## **12 k vecinos más cercanos**

---

### **In this chapter**

#### **En este capítulo**

- You learn to build a classification system using the k-nearest neighbors algorithm.  
Aprenderá a construir un sistema de clasificación utilizando el algoritmo de k vecinos más cercanos.
- You learn about feature extraction.  
Aprenderá sobre la extracción de funciones.
- You learn about regression: predicting a number, like the value of a stock tomorrow or how much a user will enjoy a movie.  
Aprende sobre regresión: predecir un número, como el valor de una acción mañana o cuánto disfrutará un usuario de una película.
- You learn about the use cases and limitations of k-nearest neighbors.  
Aprenderá sobre los casos de uso y las limitaciones de los k vecinos más cercanos.

## ***Classifying oranges vs. grapefruit***

### ***Clasificación de naranjas y pomelos***

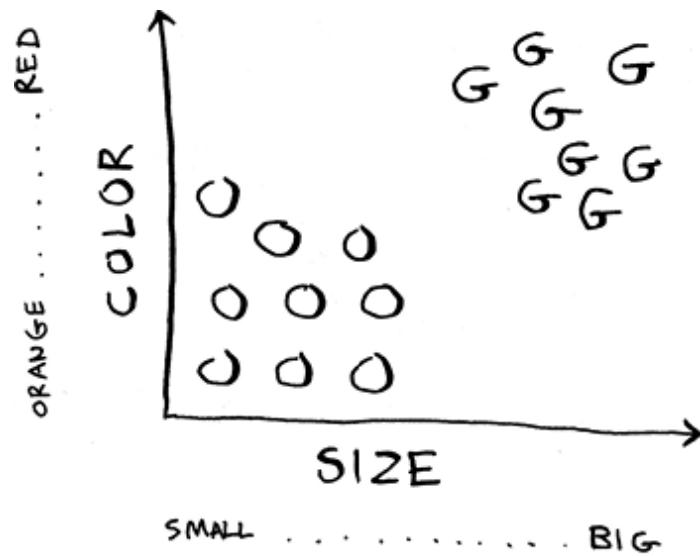
Look at this fruit. Is it an orange or a grapefruit? Well, I know that grapefruits are generally bigger and redder.

Mira esta fruta. ¿Es una naranja o un pomelo? Bueno, sé que los pomelos generalmente son más grandes y rojos.



My thought process is something like this: I have a graph in my mind.

Mi proceso de pensamiento es algo como esto: tengo un gráfico en mi mente.

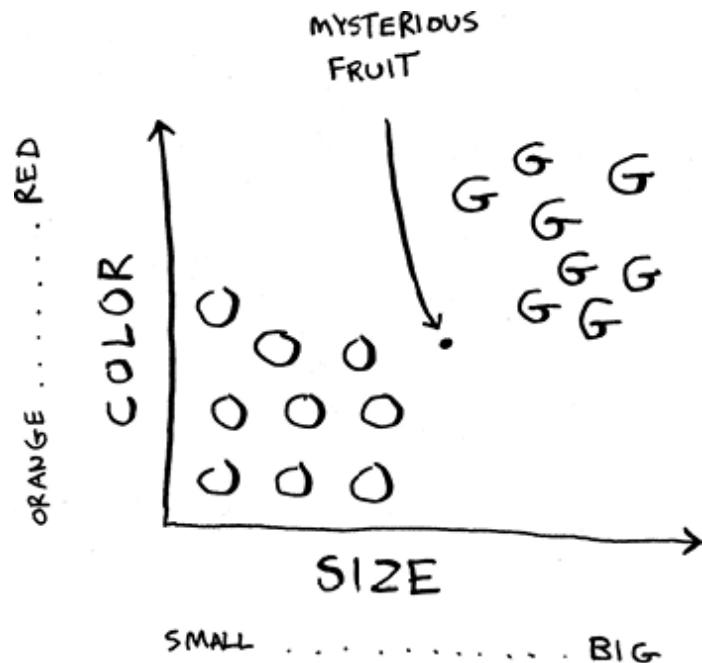


O = ORANGE

G = GRAPEFRUIT

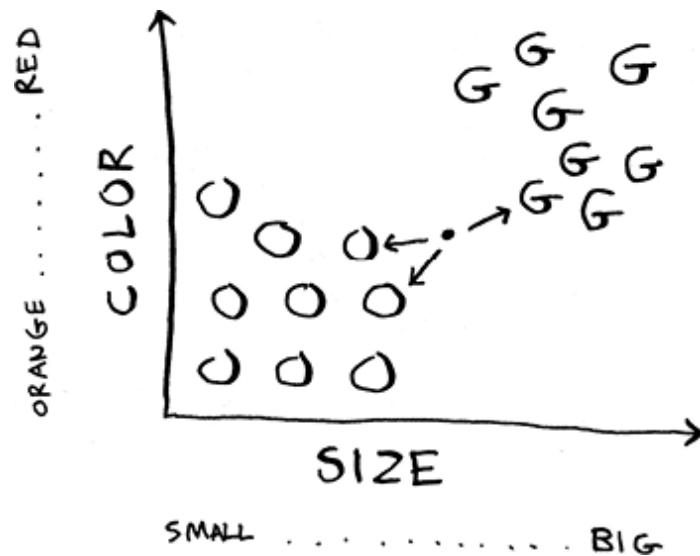
Generally speaking, the bigger, redder fruit are grapefruits. This fruit is big and red, so it's probably a grapefruit. But what if you get a fruit like this?

En general, las frutas más grandes y rojas son los pomelos. Esta fruta es grande y roja, por lo que probablemente sea una toronja. ¿Pero qué pasa si obtienes una fruta como esta?



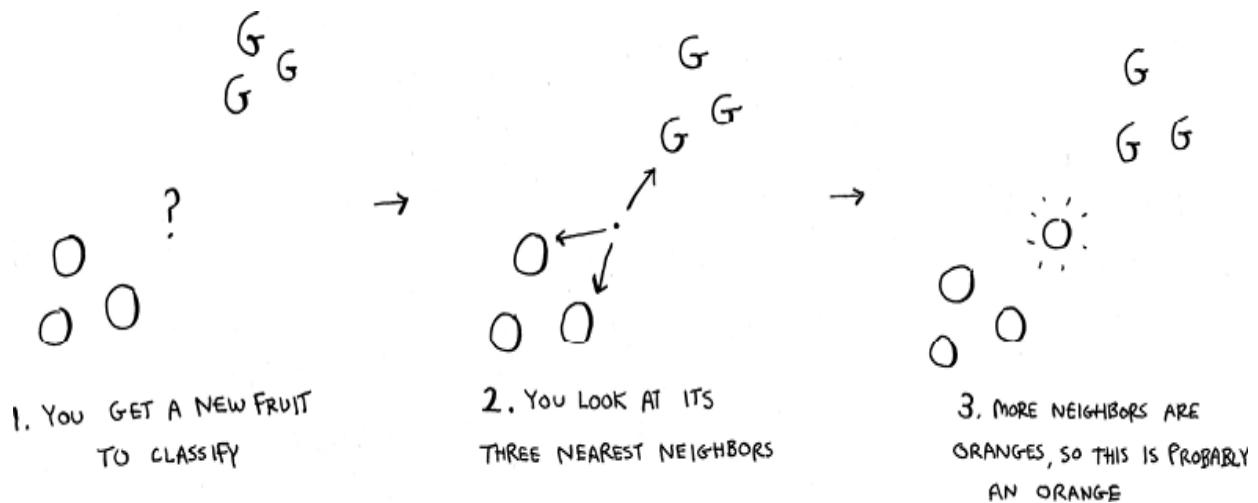
How would you *classify* this fruit? One way is to look at the neighbors of this spot. Take a look at the three closest neighbors of this spot.

¿Cómo clasificarías esta fruta? Una forma es mirar a los vecinos de este lugar. Echa un vistazo a los tres vecinos más cercanos a este lugar.



More neighbors are oranges than grapefruit. So this fruit is probably an orange. Congratulations! You just used the *k-nearest neighbors* (KNN) algorithm for *classification*! The whole algorithm is pretty simple.

Más vecinos son naranjas que pomelos. Entonces esta fruta probablemente sea una naranja. ¡Felicidades! ¡Acabas de utilizar el algoritmo de k-vecinos más cercanos (KNN) para la clasificación! Todo el algoritmo es bastante simple.



The KNN algorithm is simple but useful! If you're trying to classify something, you might want to try KNN first. Let's look at a more real-world example.

¡El algoritmo KNN es simple pero útil! Si está intentando clasificar algo, es posible que desee probar KNN primero. Veamos un ejemplo más real.

## ***Building a recommendations system***

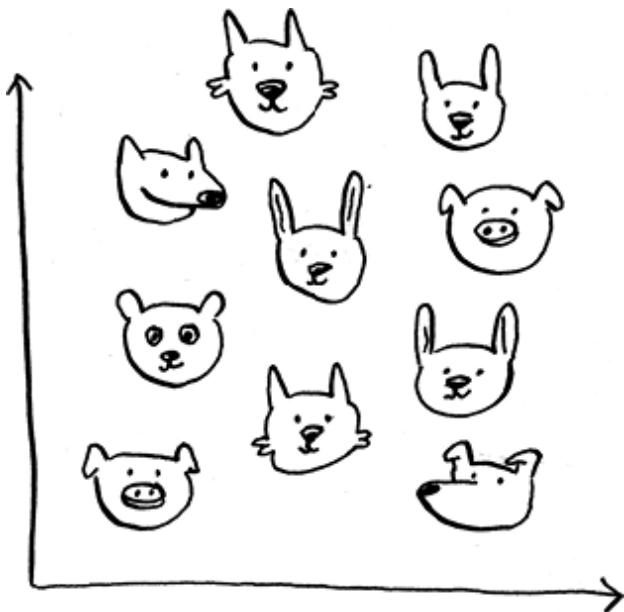
### ***Construyendo un sistema de recomendaciones***

Suppose you're Netflix, and you want to build a movie recommendations system for your users. On a high level, this is similar to the grapefruit problem!

Suponga que es Netflix y desea crear un sistema de recomendación de películas para sus usuarios. ¡En un nivel alto, esto es similar al problema de la toronja!

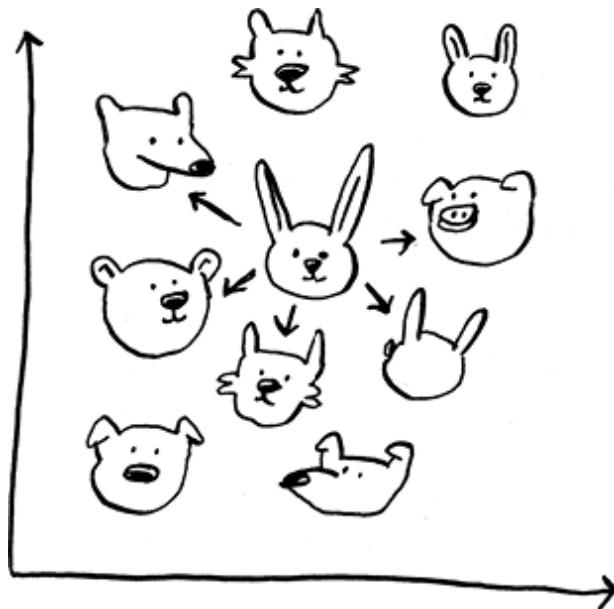
You can plot every user on a graph.

Puede trazar a cada usuario en un gráfico.



These users are plotted by similarity, so users with similar taste are plotted closer together. Suppose you want to recommend movies for Priyanka. Find the five users closest to her.

Estos usuarios se trazan por similitud, por lo que los usuarios con gustos similares se trazan más juntos. Supongamos que quieres recomendar películas para Priyanka. Encuentra a los cinco usuarios más cercanos a ella.



Justin, JC, Joey, Lance, and Chris all have similar taste in movies. So whatever movies *they* like, Priyanka will probably like too!

Justin, JC, Joey, Lance y Chris tienen gustos cinematográficos similares. Entonces, sean cuales sean las películas que les gusten, ia Priyanka probablemente también les gustarán!

Once you have this graph, building a recommendations system is easy. If Justin likes a movie, recommend it to Priyanka.

Una vez que tenga este gráfico, crear un sistema de recomendaciones es fácil. Si a Justin le gusta una película, recomiéndela a Priyanka.



But there's still a big piece missing. You graphed the users by similarity. How do you figure out how similar two users are?

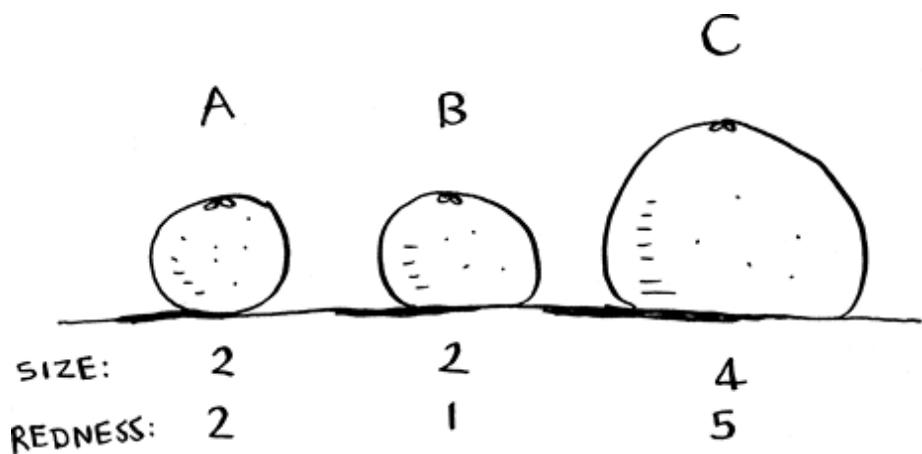
Pero todavía falta una gran pieza. Graficaste a los usuarios por similitud. ¿Cómo se puede saber qué tan similares son dos usuarios?

## Feature extraction

### Extracción de características

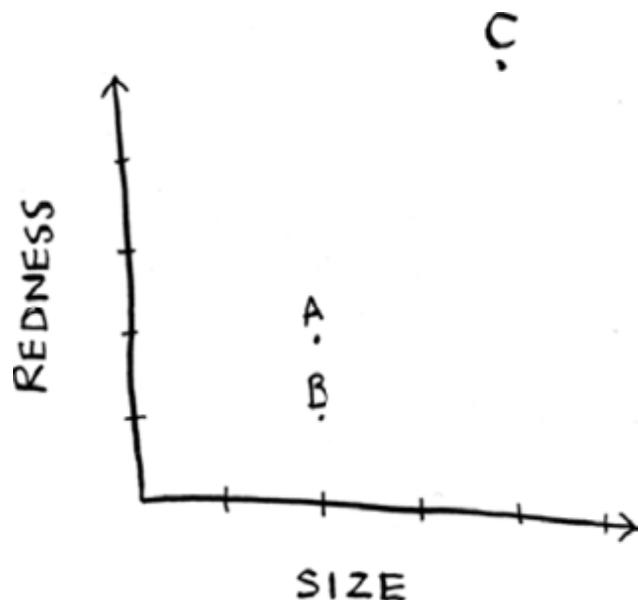
In the grapefruit example, you compared fruit based on how big they are and how red they are. Size and color are the *features* you're comparing. Now suppose you have three fruit. You can extract the features.

En el ejemplo del pomelo, comparaste las frutas según su tamaño y su color rojo. El tamaño y el color son las características que estás comparando. Ahora supongamos que tienes tres frutas. Puedes extraer las características.



Then you can graph the three fruit.

Luego puedes graficar las tres frutas.



From the graph, you can tell visually that fruits A and B are similar. Let's measure how close they are. To find the distance between two points, you use the Pythagorean formula.

En la gráfica puedes ver visualmente que las frutas A y B son similares. Midamos qué tan cerca están. Para encontrar la distancia entre dos puntos se utiliza la fórmula pitagórica.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

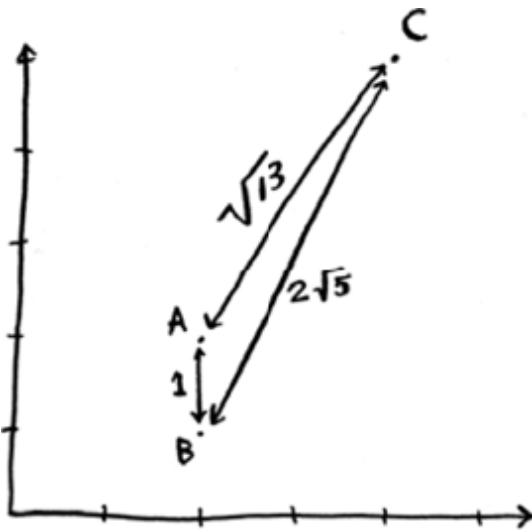
Here's the distance between A and B, for example.

Aquí está la distancia entre A y B, por ejemplo.

$$\begin{aligned} & \sqrt{(2-2)^2 + (2-1)^2} \\ & = \sqrt{0 + 1} \\ & = \sqrt{1} \\ & = 1 \end{aligned}$$

The distance between A and B is 1. You can find the rest of the distances, too.

La distancia entre A y B es 1. También puedes encontrar el resto de las distancias.

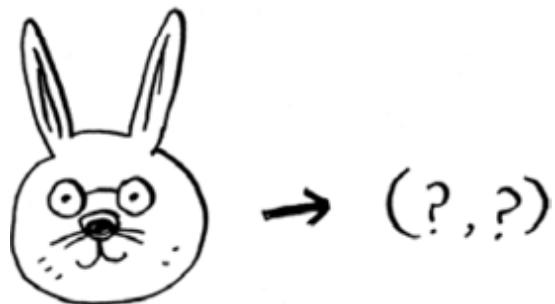


The distance formula confirms what you saw visually: fruits A and B are similar.

La fórmula de la distancia confirma lo que viste visualmente: las frutas A y B son similares.

Suppose you're comparing Netflix users instead. You need some way to graph the users. So, you need to convert each user to a set of coordinates, just as you did for fruit.

Supongamos que estás comparando usuarios de Netflix. Necesita alguna forma de graficar a los usuarios. Por lo tanto, debes convertir a cada usuario a un conjunto de coordenadas, tal como lo hiciste con la fruta.



Once you can graph users, you can measure the distance between them.

Una vez que puedas representar gráficamente a los usuarios, podrás medir la distancia entre ellos.

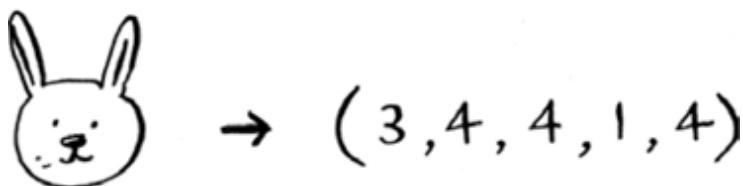
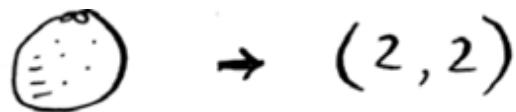
Here's how you can convert users into a set of numbers. When users sign up for Netflix, have them rate some categories of movies based on how much they like those categories. For each user, you now have a set of ratings!

Así es como puedes convertir usuarios en un conjunto de números. Cuando los usuarios se registren en Netflix, pídale que califiquen algunas categorías de películas en función de cuánto les gustan esas categorías. ¡Para cada usuario, ahora tienes un conjunto de calificaciones!

			
	PRIYANKA	JUSTIN	MORPHEUS
COMEDY	3	4	2
ACTION	4	3	5
DRAMA	4	5	1
HORROR	1	1	3
ROMANCE	4	5	1

Priyanka and Justin like Romance and hate Horror. Morpheus likes Action but hates Romance (he hates it when a good action movie gets ruined by a cheesy romantic scene). Remember how, in oranges versus grapefruit, each fruit was represented by a set of two numbers? Here, each user is represented by a set of five numbers.

A Priyanka y Justin les gusta el romance y odian el terror. A Morfeo le gusta la acción pero odia el romance (odia cuando una buena película de acción se arruina por una escena romántica cursi). ¿Recuerdas cómo, en naranjas versus pomelos, cada fruta estaba representada por un conjunto de dos números? Aquí, cada usuario está representado por un conjunto de cinco números.



A mathematician would say instead of calculating the distance in two dimensions, you're now calculating the distance in *five* dimensions. But the distance formula remains the same.

Un matemático diría que en lugar de calcular la distancia en dos dimensiones, ahora estás calculando la distancia en cinco dimensiones. Pero la fórmula de la distancia sigue siendo la misma.

$$\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2 + (c_1 - c_2)^2 + (d_1 - d_2)^2 + (e_1 - e_2)^2}$$

It just involves a set of five numbers instead of a set of two numbers.

Simplemente implica un conjunto de cinco números en lugar de un conjunto de dos números.

The distance formula is flexible: you could have a set of a *million* numbers and still use the same old distance formula to find the distance. Maybe you're wondering, "What does

*distance* mean when you have five numbers?" The distance tells you how similar those sets of numbers are.

La fórmula de la distancia es flexible: podrías tener un conjunto de un millón de números y seguir usando la misma fórmula de distancia antigua para encontrar la distancia. Quizás te estés preguntando: "¿Qué significa la distancia cuando tienes cinco números?" La distancia te dice qué tan similares son esos conjuntos de números.

$$\begin{aligned} & \sqrt{(3-4)^2 + (4-3)^2 + (4-5)^2 + (1-1)^2 + (4-5)^2} \\ &= \sqrt{1 + 1 + 1 + 0 + 1} \\ &= \sqrt{4} \\ &= 2 \end{aligned}$$

Here's the distance between Priyanka and Justin.

Aquí está la distancia entre Priyanka y Justin.

**NOTE** By the way, here's a bit of terminology you'll see often. Those arrays of numbers like (2, 2) for the grapefruit or (3, 4, 4, 1, 4) for Priyanka's taste in movies are called *vectors*. So if you're reading a paper on machine learning, and you see the authors talking about vectors, they mean an array of numbers like that.

Nota Por cierto, aquí hay un poco de terminología que verá con frecuencia. Esas matrices de números como (2, 2) para la toronja o (3, 4, 4, 1, 4) para el gusto de Priyanka por las películas se llaman vectores. Entonces, si estás leyendo un artículo sobre aprendizaje automático y ves a los autores hablando de vectores, se refieren a una serie de números como ese.

Priyanka and Justin are pretty similar. What's the difference between Priyanka and Morpheus? Calculate the distance before moving on.

Priyanka y Justin son bastante similares. ¿Cuál es la diferencia entre Priyanka y Morfeo? Calcula la distancia antes de continuar.

Did you get it right? Priyanka and Morpheus  $\sqrt{24}$  are apart. The distance tells you that Priyanka's tastes are more like Justin's than Morpheus's.

¿Lo entendiste correctamente? Priyanka y Morfeo  $\sqrt{24}$  están separados. La distancia te dice que los gustos de Priyanka se parecen más a los de Justin que a los de Morfeo.

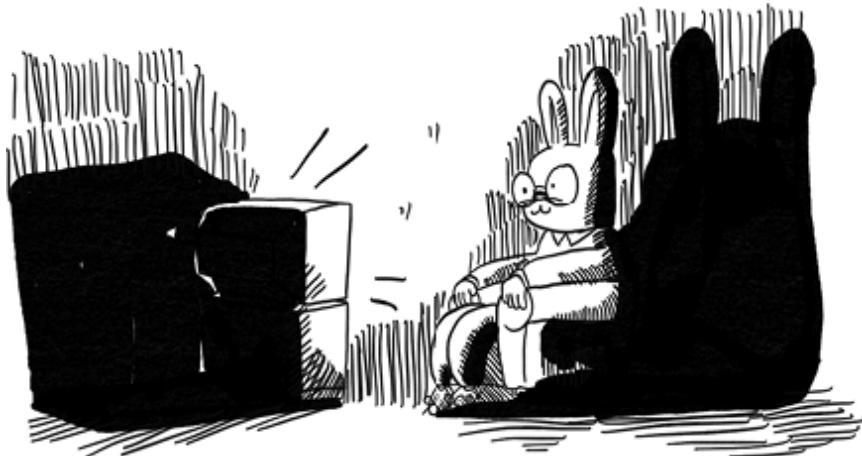
Great! Now recommending movies to Priyanka is easy: if Justin likes a movie, recommend it to Priyanka, and vice versa. You just built a movie recommendations system!

¡Excelente! Ahora recomendar películas a Priyanka es fácil: si a Justin le gusta una película, recomiéndela a Priyanka y viceversa. ¡Acabas de crear un sistema de recomendaciones de películas!

If you're a Netflix user, Netflix will keep telling you, "Please rate more movies. The more movies you rate, the better your recommendations will be." Now you know why. The more movies you rate, the more accurately Netflix can see what other users you're similar to.

Si es usuario de Netflix, Netflix le seguirá diciendo: "Califique más películas. Cuantas más películas califiques,

mejores serán tus recomendaciones". Ahora sabes por qué. Cuantas más películas califiques, Netflix podrá ver con mayor precisión a qué otros usuarios te pareces.



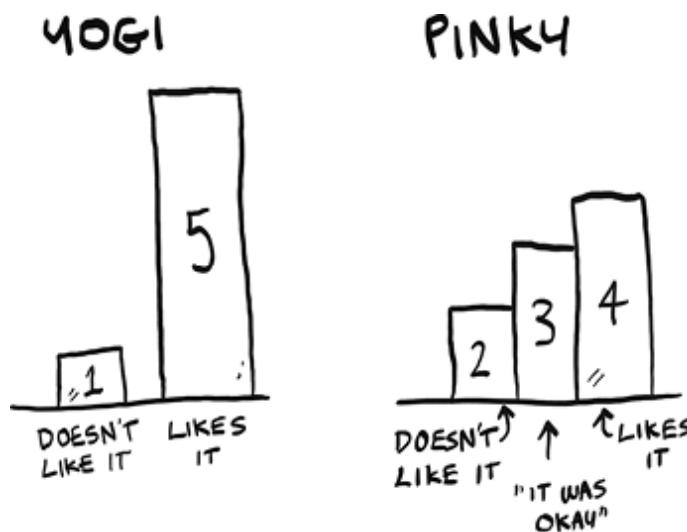
## EXERCISES

### EJERCICIOS

**12.1** In the Netflix example, you calculated the distance between two different users using the distance formula. But not all users rate movies the same way. Suppose you have two users, Yogi and Pinky, who have the same taste in movies. But Yogi rates any movie he likes as a 5, whereas Pinky is choosier and reserves the 5s for only the best. They're well matched, but according to the distance algorithm, they aren't neighbors. How would you take their different rating strategies into account?

12.1 En el ejemplo de Netflix, calculó la distancia entre dos usuarios diferentes usando la fórmula de distancia. Pero no todos los usuarios valoran las

películas de la misma manera. Supongamos que tienes dos usuarios, Yogi y Pinky, que tienen el mismo gusto cinematográfico. Pero Yogi califica cualquier película que le guste con un 5, mientras que Pinky es más exigente y reserva el 5 sólo para las mejores. Están bien emparejados, pero según el algoritmo de distancia, no son vecinos. ¿Cómo tomaría en cuenta sus diferentes estrategias de calificación?



**12.2** Suppose Netflix nominates a group of "influencers." For example, Quentin Tarantino and Wes Anderson are influencers on Netflix, so their ratings count for more than a normal user's. How would you change the recommendations system so it's biased toward the ratings of influencers?

12.2 Supongamos que Netflix nomina a un grupo de "influencers". Por ejemplo, Quentin Tarantino y Wes Anderson son personas influyentes en Netflix, por lo

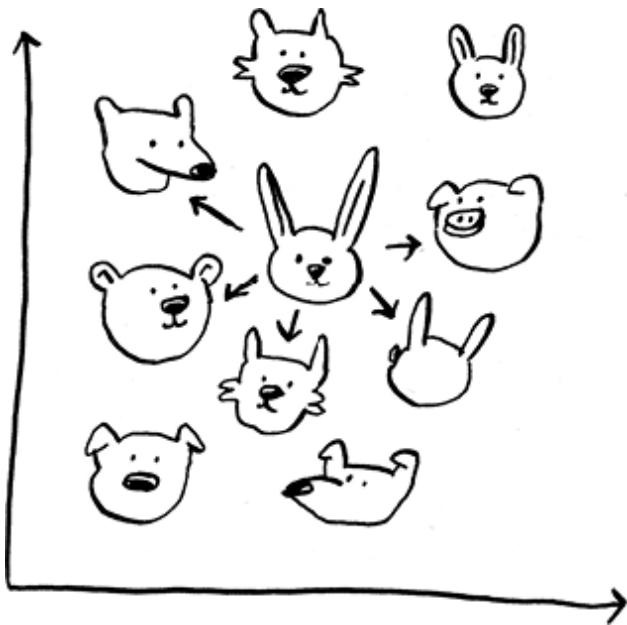
que sus calificaciones cuentan más que las de un usuario normal. ¿Cómo cambiarías el sistema de recomendaciones para que esté sesgado hacia las calificaciones de los influencers?

## ***Regression***

## ***Regresión***

Suppose you want to do more than just recommend a movie: you want to guess how Priyanka will rate this movie. Take the five people closest to her.

Supongamos que quiere hacer algo más que simplemente recomendar una película: quiere adivinar cómo calificará Priyanka esta película. Toma a las cinco personas más cercanas a ella.



By the way, I keep talking about the closest five people. There's nothing special about the number 5: you could do the closest 2, or 10, or 10,000. That's why the algorithm is called k-nearest neighbors and not five-nearest neighbors!

Por cierto, sigo hablando de las cinco personas más cercanas. No hay nada especial en el número 5: puedes hacer los 2, 10 o 10 000 más cercanos. ¡Es por eso que el algoritmo se llama k-vecinos más cercanos y no cinco vecinos más cercanos!

Suppose you're trying to guess a rating for *Pitch Perfect*. Well, how did Justin, JC, Joey, Lance, and Chris rate it?

Suponga que está intentando adivinar una calificación para *Pitch Perfect*. Bueno, ¿cómo lo calificaron Justin, JC, Joey, Lance y Chris?

JUSTIN : 5  
JC : 4  
JOEY : 4  
LANC E : 5  
CHRIS : 3

You could take the average of their ratings and get 4.2 stars. That's called *regression*. These are the two basic things you'll do with KNN—classification and regression:

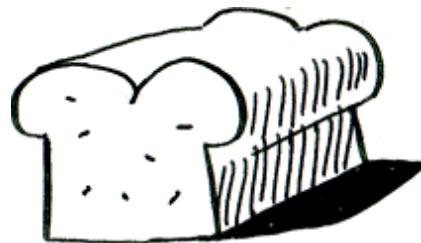
Podrías tomar el promedio de sus calificaciones y obtener 4,2 estrellas. Eso se llama regresión. Estas son las dos cosas básicas que hará con KNN: clasificación y regresión:

- Classification = categorization into a group  
Clasificación = categorización en un grupo
- Regression = predicting a response (like a number)  
Regresión = predecir una respuesta (como un número)

Regression is very useful. Suppose you run a small bakery in Berkeley, and you make fresh bread every day. You're trying to predict how many loaves to make for today. You have a set of features:

La regresión es muy útil. Supongamos que tiene una pequeña panadería en Berkeley y elabora pan fresco todos los días. Estás tratando de predecir cuántos panes prepararás para hoy. Tienes un conjunto de características:

- Weather on a scale of 1 to 5 (1 = bad, 5 = great).  
Clima en una escala del 1 al 5 (1 = malo, 5 = excelente).
- Weekend or holiday? (1 if it's a weekend or a holiday, 0 otherwise.)  
¿Fin de semana o feriado? (1 si es fin de semana o feriado, 0 en caso contrario).
- Is there a game on? (1 if yes, 0 if no.)  
¿Hay un juego en marcha? (1 si es así, 0 si no).



And you know how many loaves of bread you've sold in the past for different sets of features.

Y sabrá cuántas hogazas de pan vendió en el pasado para diferentes conjuntos de características.

$$\boxed{A.} (5, 1, \emptyset) = \underset{\text{LOAVES}}{3\phi\phi} \quad \boxed{B.} (3, 1, 1) = \underset{\text{LOAVES}}{225}$$

$$\boxed{C.} (1, 1, \emptyset) = \underset{\text{LOAVES}}{75} \quad \boxed{D.} (4, \emptyset, 1) = \underset{\text{LOAVES}}{2\phi\phi}$$

$$\boxed{E.} (4, \emptyset, \emptyset) = \underset{\text{LOAVES}}{15\phi} \quad \boxed{F.} (2, \emptyset, \emptyset) = \underset{\text{LOAVES}}{5\phi}$$

Today is a weekend day with good weather. Based on the data you just saw, how many loaves will you sell? Let's use KNN, where  $k = 4$ . First, figure out the four nearest neighbors for this point.

Hoy es un día de fin de semana con buen tiempo. Según los datos que acabas de ver, ¿cuántos panes venderás? Usemos KNN, donde  $k = 4$ . Primero, calcule los cuatro vecinos más cercanos para este punto.

$$(4, 1, \emptyset) = ?$$

Here are the distances. A, B, D, and E are the closest.

Aquí están las distancias. A, B, D y E son los más cercanos.

- A. 1 ←
- B. 2 ←
- C. 9
- D. 2 ←
- E. 1 ←
- F. 5

Take an average of the loaves sold on those days, and you get 218.75. That's how many loaves you should make for today!

Tome un promedio de los panes vendidos en esos días y obtendrá 218,75. ¡Esa es la cantidad de panes que debes hacer hoy!

## Cosine similarity

### Similitud del coseno

So far, you've been using the distance formula to compare the distance between two users. Is this the best formula to use? A common one used in practice is *cosine similarity*. Suppose two users are similar, but one is more conservative in their ratings. They both loved Manmohan Desai's *Amar Akbar Anthony*. Paul rated it 5 stars, but Rowan rated it 4 stars. If you keep using the distance formula, these two users might not be each other's neighbors, even though they have similar taste.

Hasta ahora, has estado usando la fórmula de la distancia para comparar la distancia entre dos usuarios. ¿Es esta la mejor fórmula para usar? Uno de los más utilizados en la práctica es la similitud del coseno. Supongamos que dos usuarios son similares, pero uno es más conservador en sus calificaciones. Ambos amaban a Amar Akbar Anthony de Manmohan

Desai. Paul lo calificó con 5 estrellas, pero Rowan lo calificó con 4 estrellas. Si sigues usando la fórmula de la distancia, es posible que estos dos usuarios no sean vecinos, aunque tengan gustos similares.

Cosine similarity doesn't measure the distance between two vectors. Instead, it compares the angles of the two vectors. It's better at dealing with cases like this. Cosine similarity is out of the scope of this book, but look it up if you use KNN!

La similitud del coseno no mide la distancia entre dos vectores. En cambio, compara los ángulos de los dos vectores. Es mejor lidiar con casos como este. La similitud del coseno está fuera del alcance de este libro, ¡pero búscalas si usas KNN!

## Picking good features

### Elegir buenas características

To figure out recommendations, you had users rate categories of movies. What if you had them rate pictures of cats instead? Then you'd find users who rated those pictures similarly. This would probably be a worse recommendations engine because the "features" don't have a lot to do with taste in movies!

Para obtener recomendaciones, los usuarios calificaban categorías de películas. ¿Qué pasaría si en su lugar les pidieran que calificaran fotografías de gatos? Luego encontrarás usuarios que calificaron esas imágenes de manera similar. ¡Este probablemente sería un peor motor de recomendaciones porque las "características" no tienen mucho que ver con el gusto por las películas!



Or suppose you ask users to rate movies so you can give them recommendations, but you only ask them to rate *Toy Story*, *Toy Story 2*, and *Toy Story 3*. This won't tell you a lot about the users' movie tastes!

O supongamos que les pide a los usuarios que califiquen películas para poder darles recomendaciones, pero solo les pide que califiquen *Toy Story*, *Toy Story 2* y *Toy Story 3*. ¡Esto no le dirá mucho sobre los gustos cinematográficos de los usuarios!

When you're working with KNN, it's really important to pick the right features to compare against. Picking the right features means

Cuando trabaja con KNN, es muy importante elegir las funciones adecuadas con las que comparar. Elegir las características adecuadas significa

- Features that directly correlate to the movies you're trying to recommend

Funciones que se correlacionan directamente con las películas que intentas recomendar

- Features that don't have a bias (for example, if you ask the users to only rate comedy movies, that doesn't tell you whether they like action movies)

Funciones que no tienen prejuicios (por ejemplo, si les pide a los usuarios que califiquen solo películas de comedia, eso no le indica si les gustan las películas de acción)

Do you think ratings are a good way to recommend movies? Maybe I rated *Inception* more highly than *Legally Blonde*, but I actually spend more time watching *Legally Blonde*. How would you improve this Netflix recommendations system?

¿Crees que las calificaciones son una buena forma de recomendar películas? Tal vez califiqué a *Inception* mejor que *Legally Blonde*, pero en realidad paso más tiempo viendo *Legally Blonde*. ¿Cómo mejorarías este sistema de recomendaciones de Netflix?

Going back to the bakery: Can you think of two good and two bad features you could have picked for the bakery? Maybe you need to make more loaves after you advertise in the paper. Or maybe you need to make more loaves on Mondays.

Volviendo a la panadería: ¿Puedes pensar en dos características buenas y dos malas que podrías haber elegido para la panadería? Tal vez necesites hacer más panes después de anunciarlo en el periódico. O tal vez necesites hacer más panes los lunes.

There's no one right answer when it comes to picking good features. You have to think about all the different things you need to consider.

No existe una respuesta correcta cuando se trata de elegir buenas funciones. Tienes que pensar en todas las diferentes cosas que debes considerar.

## EXERCISE

### EJERCICIO

**12.3** Netflix has millions of users. The earlier example looked at the five closest neighbors for building the recommendations system. Is this too low? Too high?

12.3 Netflix tiene millones de usuarios. El ejemplo anterior analizó a los cinco vecinos más cercanos para construir el sistema de recomendaciones. ¿Es esto demasiado bajo? ¿Demasiado alto?

## *Introduction to machine learning*

## *Introducción al aprendizaje automático*

KNN is a really useful algorithm, and it's your introduction to the magical world of machine learning! Machine learning is all about making your computer more intelligent. You already saw one example of machine learning: building a

recommendations system. Let's look at some other examples.

iKNN es un algoritmo realmente útil y es su introducción al mágico mundo del aprendizaje automático! El aprendizaje automático consiste en hacer que su computadora sea más inteligente. Ya has visto un ejemplo de aprendizaje automático: la creación de un sistema de recomendaciones. Veamos algunos otros ejemplos.



## OCR

## LOC

OCR stands for *optical character recognition*. It means you can take a photo of a page of text, and your computer will automatically read the text for you. Google uses OCR to digitize books. How does OCR work? For example, consider this number.

OCR significa reconocimiento óptico de caracteres. Significa que puedes tomar una foto de una página de texto y tu computadora leerá automáticamente el texto por ti. Google utiliza OCR para digitalizar libros. ¿Cómo funciona el OCR? Por ejemplo, considere este número.



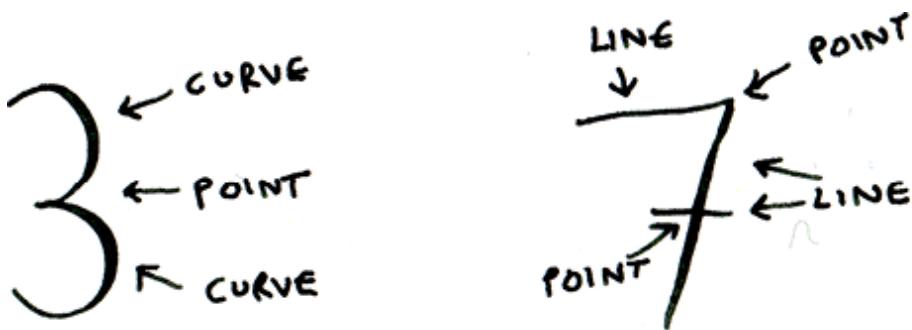
How would you automatically figure out what number this is? You can use KNN for this:

¿Cómo sabrías automáticamente qué número es este?  
Puedes usar KNN para esto:

1. Go through a lot of images of numbers and extract features of those numbers.  
Revise muchas imágenes de números y extraiga características de esos números.
2. When you get a new image, extract the features of that image and see what its nearest neighbors are!  
Cuando obtenga una nueva imagen, extraiga las características de esa imagen y vea cuáles son sus vecinos más cercanos.

It's the same problem as oranges versus grapefruit.  
Generally speaking, OCR algorithms measure lines, points, and curves.

Es el mismo problema que el de las naranjas versus el pomelo. En términos generales, los algoritmos de OCR miden líneas, puntos y curvas.



Then, when you get a new character, you can extract the same features from it.

Luego, cuando obtengas un nuevo personaje, podrás extraerle las mismas características.

Feature extraction is a lot more complicated in OCR than the fruit example. But it's important to understand that even complex technologies build on simple ideas, like KNN. You could use the same ideas for speech recognition or face recognition. When you upload a photo to Facebook, sometimes it's smart enough to tag people in the photo automatically. That's machine learning in action!

La extracción de características es mucho más complicada en OCR que en el ejemplo de la fruta. Pero es importante comprender que incluso las tecnologías complejas se basan en ideas simples, como KNN. Podrías utilizar las mismas ideas para el reconocimiento de voz o el reconocimiento facial. Cuando subes una foto a Facebook, a veces es lo

suficientemente inteligente como para etiquetar personas en la foto automáticamente. ¡Eso es aprendizaje automático en acción!

The first step of OCR, where you go through images of numbers and extract features, is called—wait for it—*feature extraction*. This is where you transform the images into something that your machine learning algorithm can work with. The next step is called *training*, where you train a model on your features so it can recognize numbers in images. Most machine-learning algorithms have a training step: before your computer can do the task, it must be trained. The next example involves spam filters, and it has a training step.

El primer paso del OCR, en el que se revisan imágenes de números y se extraen características, se llama (espérelo) extracción de características. Aquí es donde transformas las imágenes en algo con lo que tu algoritmo de aprendizaje automático pueda funcionar. El siguiente paso se llama entrenamiento, donde entrenas un modelo según tus características para que pueda reconocer números en imágenes. La mayoría de los algoritmos de aprendizaje automático tienen un paso de entrenamiento: antes de que su computadora pueda realizar la tarea, debe ser entrenada. El siguiente ejemplo involucra filtros de spam y tiene un paso de capacitación.

## Building a spam filter

### Construyendo un filtro de spam

Spam filters use another simple algorithm called the *Naive Bayes classifier*. First, you train your Naive Bayes classifier on some data.

Los filtros de spam utilizan otro algoritmo simple llamado clasificador Naive Bayes. Primero, entrena su clasificador Naive Bayes con algunos datos.

SUBJECT	SPAM?
"RESET YOUR PASSWORD"	NOT SPAM
"YOU HAVE WON 1 MILLION DOLLARS"	SPAM
"SEND ME YOUR PASSWORD"	SPAM
"HAPPY BIRTHDAY"	NOT SPAM

Suppose you get an email with the subject "Collect your million dollars now!" Is it spam? You can break this sentence into words. Then, for each word, see what the probability is for that word to show up in a spam email. For example, in this very simple model, the word *million* only appears in spam emails. Naive Bayes figures out the probability that something is likely to be spam. It has applications similar to KNN. For example, you could use Naive Bayes to categorize

fruit: you have a fruit that's big and red. What's the probability that it's a grapefruit? It's another simple algorithm that's fairly effective. We love those algorithms!

Supongamos que recibe un correo electrónico con el asunto "¡Recoja su millón de dólares ahora!" ¿Es spam? Puedes dividir esta oración en palabras. Luego, para cada palabra, vea cuál es la probabilidad de que esa palabra aparezca en un correo electrónico no deseado. Por ejemplo, en este modelo tan simple, la palabra millón sólo aparece en los correos electrónicos no deseados. Naive Bayes calcula la probabilidad de que algo sea spam. Tiene aplicaciones similares a KNN. Por ejemplo, podrías usar Naive Bayes para categorizar frutas: tienes una fruta que es grande y roja. ¿Cuál es la probabilidad de que sea una toronja? Es otro algoritmo simple que es bastante efectivo. ¡Nos encantan esos algoritmos!

## Predicting the stock market

### Predecir el mercado de valores

Here's something that's hard to do with machine learning: predicting whether the stock market will go up or down. How do you pick good features in a stock market? Suppose you say that if the stock went up yesterday, it will go up today. Is that a good feature? Or suppose you say that the stock will always go down in May. Will that work? There's no guaranteed way to use past numbers to predict future

performance. Predicting the future is hard, and it's almost impossible when there are so many variables involved.

Aquí hay algo que es difícil de hacer con el aprendizaje automático: predecir si el mercado de valores subirá o bajará. ¿Cómo se eligen buenas características en un mercado de valores? Supongamos que si la acción subió ayer, subirá hoy. ¿Es esa una buena característica? O supongamos que dice que las acciones siempre bajarán en mayo. ¿Eso funcionará? No existe una forma garantizada de utilizar números pasados para predecir el rendimiento futuro. Predecir el futuro es difícil y casi imposible cuando hay tantas variables involucradas.



## ***A high-level overview of training an ML model***

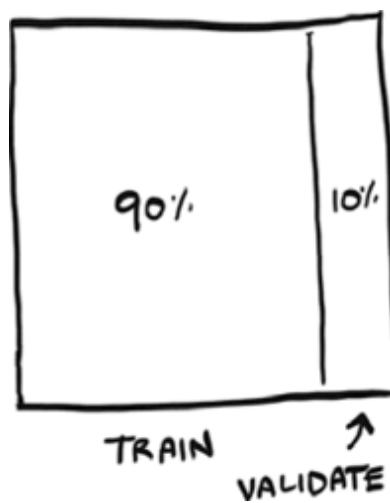
***Una descripción general de alto nivel sobre el entrenamiento de un modelo de ML***



Now that we've seen a few examples, let's look at the steps to training an ML model. First gather the data. In the Netflix example, our data was movie ratings from users. Then you need to clean the data. Cleaning means getting rid of bad data. For example, you may have users who don't like being prompted to rate movies, so they rate movies randomly and

move to the next screen. You will want to remove this data from your data set. You will then need to extract features from your data.

Ahora que hemos visto algunos ejemplos, veamos los pasos para entrenar un modelo de ML. Primero recopile los datos. En el ejemplo de Netflix, nuestros datos fueron las calificaciones de películas de los usuarios. Entonces necesitas limpiar los datos. Limpiar significa deshacerse de los datos incorrectos. Por ejemplo, es posible que haya usuarios a quienes no les gusta que se les solicite que califiquen películas, por lo que las califican al azar y pasan a la siguiente pantalla. Querrá eliminar estos datos de su conjunto de datos. Luego deberá extraer características de sus datos.



After you have your features, it's time to train your model. Select a model like KNN, SVM's, or a neural network and train it with 90% of your data. Keep the remaining 10% to validate the model. After your model is trained, you will test

it by asking it to make a prediction. You can use that 10% of data to see how good that prediction is.

Una vez que tenga sus funciones, es hora de entrenar su modelo. Seleccione un modelo como KNN, SVM o una red neuronal y entrénelo con el 90% de sus datos. Conserve el 10% restante para validar el modelo. Una vez entrenado su modelo, lo probará pidiéndole que haga una predicción. Puedes usar ese 10% de los datos para ver qué tan buena es esa predicción.

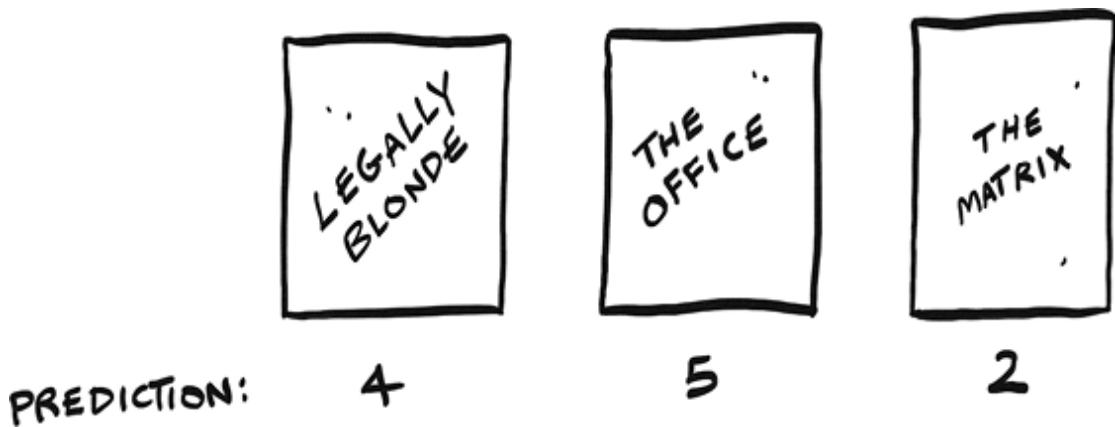
For example, let's say we want to test the Netflix recommendations model. We can ask it how Priyanka would like these movies and shows:

Por ejemplo, digamos que queremos probar el modelo de recomendaciones de Netflix. Podemos preguntarle cómo le gustaría a Priyanka estas películas y programas:



Our model comes back with its predictions.

Nuestro modelo regresa con sus predicciones.



We know which movies Priyanka likes -- that was part of the 10% of data we held back. We can compare that against the model's predictions.

Sabemos qué películas le gustan a Priyanka; eso fue parte del 10% de los datos que retuvimos. Podemos comparar eso con las predicciones del modelo.



Not bad! In that case we can say the model made a good prediction, because the numbers are pretty close to Priyanka's actual ratings. This step of testing the model is called validating or evaluating the model.

iNada mal! En ese caso, podemos decir que el modelo hizo una buena predicción, porque las cifras son bastante cercanas a las calificaciones reales de Priyanka. Este paso de probar el modelo se llama validar o evaluar el modelo.

After evaluating the model, we may want to go back and adjust it. For example, let's say we've built a KNN model where  $K = \text{five}$ . We may want to try it with  $K = \text{seven}$  to see if it gives better results. This is called *parameter tuning*.

Después de evaluar el modelo, es posible que queramos regresar y ajustarlo. Por ejemplo, digamos que hemos creado un modelo KNN donde  $K = \text{cinco}$ . Quizás queramos probarlo con  $K = \text{siete}$  para ver si da mejores resultados. Esto se llama ajuste de parámetros.

After you're done training and evaluating the model, you have a model ready to go! These are the steps in building an ML model at a high level.

Una vez que haya terminado de entrenar y evaluar el modelo, tendrá un modelo listo para usar! Estos son los pasos para construir un modelo de ML de alto nivel.

## **Recap**

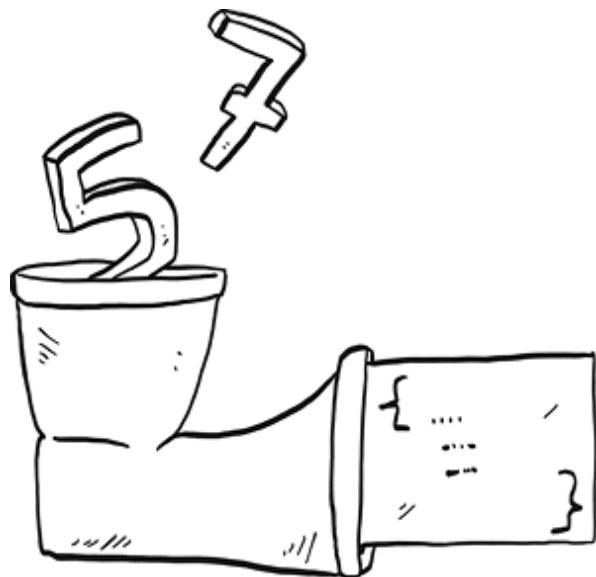
## **Resumen**

I hope this gives you an idea of all the different things you can do with KNN and with machine learning! Machine

learning is an interesting field that you can go pretty deep into if you decide to.

¡Espero que esto te dé una idea de todas las cosas diferentes que puedes hacer con KNN y con el aprendizaje automático! El aprendizaje automático es un campo interesante en el que puedes profundizar bastante si así lo decides.

- KNN is used for classification and regression and involves looking at the k-nearest neighbors.  
KNN se utiliza para clasificación y regresión e implica observar los k vecinos más cercanos.
- Classification = categorization into a group.  
Clasificación = categorización en un grupo.
- Regression = predicting a response (like a number).  
Regresión = predecir una respuesta (como un número).
- Feature extraction means converting an item (like a fruit or a user) into a list of numbers that can be compared.  
La extracción de características significa convertir un elemento (como una fruta o un usuario) en una lista de números que se pueden comparar.
- Picking good features is an important part of a successful KNN algorithm.  
Elegir buenas características es una parte importante del éxito de un algoritmo KNN.



FEATURE EXTRACTION

## ***inside back cover***

---

## ***interior de la contraportada***

---



## **13 Where to go next**

---

## **13 Adónde ir a continuación**

---

### **In this chapter**

#### **En este capítulo**

- You get a brief overview of 10 algorithms that haven't been covered in this book and why they're useful.

Obtendrá una breve descripción general de 10 algoritmos que no se han tratado en este libro y por qué son útiles.

- You get pointers on what to read next, depending on what your interests are.

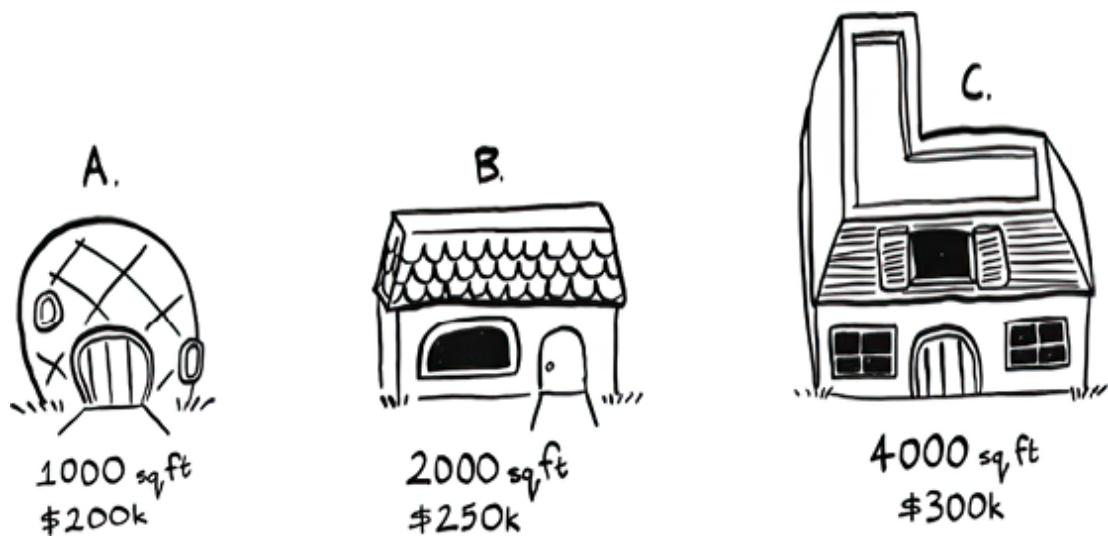
Obtendrá sugerencias sobre qué leer a continuación, según cuáles sean sus intereses.

### ***Linear regression***

### ***Regresión lineal***

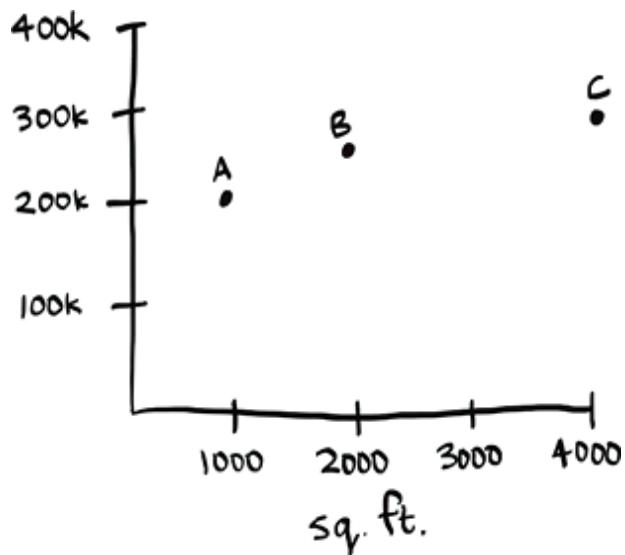
Suppose you need to sell your house. It is 3,000 ft<sup>2</sup>. You look at the homes recently sold in your neighborhood.

Supongamos que necesita vender su casa. Tiene 3.000 pies<sup>2</sup>. Mira las casas vendidas recientemente en tu vecindario.



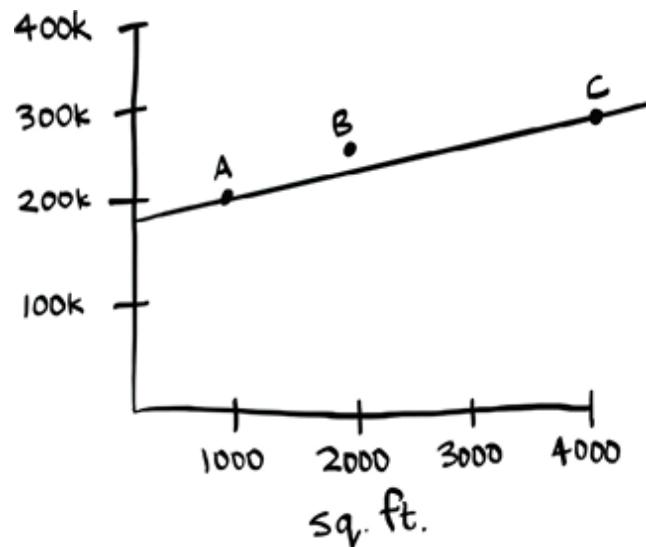
Based on this information, how would you price your house?  
Here's one way you could do it. Plot all the points.

Con base en esta información, ¿cómo pondría el precio a su casa? Aquí tienes una forma de hacerlo. Traza todos los puntos.



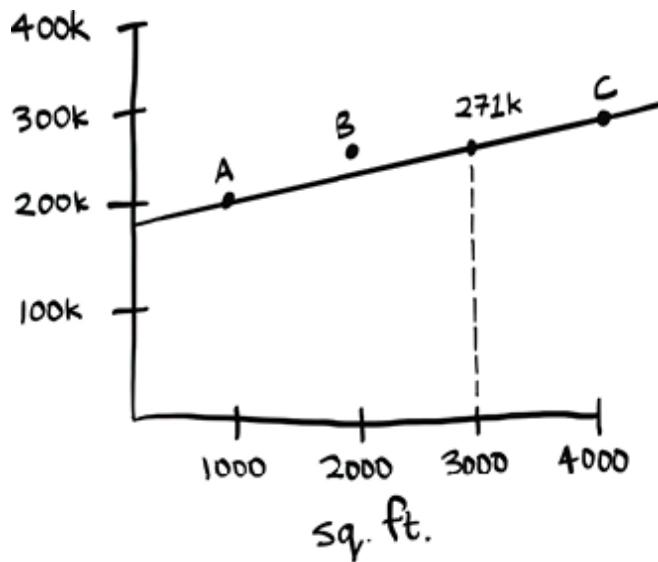
Then eyeball a line through these points.

Luego, traza una línea que pase por estos puntos.



Now you can see where  $3000 \text{ ft}^2$  lands on that line, and that would be a pretty good starting price for your home:

Ahora puede ver dónde terminan  $3000 \text{ pies}^2$  en esa línea, y ese sería un precio inicial bastante bueno para su casa:



This is how linear regression works. Given a bunch of points, it tries to fit a line to them, and then you can use that line to make predictions.

Así es como funciona la regresión lineal. Dado un conjunto de puntos, intenta ajustarles una línea y luego puedes usar esa línea para hacer predicciones.

Linear regression has been used in statistics for a long time, and now it is being widely used in machine learning because it is an easy first technique to try. It is useful if your values are continuous. If you are trying to predict something, linear regression might be a good place to start.

La regresión lineal se ha utilizado en estadística durante mucho tiempo y ahora se está utilizando ampliamente en el aprendizaje automático porque es una primera técnica fácil de probar. Es útil si sus valores son continuos. Si intenta predecir algo, la regresión lineal podría ser un buen punto de partida.

## **Inverted indexes**

### **índices invertidos**

Here's a very simplified version of how a search engine works. Suppose you have three web pages with this simple content.

Aquí tienes una versión muy simplificada de cómo funciona un motor de búsqueda. Supongamos que tiene tres páginas web con este contenido simple.



Let's build a hash table from this content.

Construyamos una tabla hash a partir de este contenido.

HI	A, B
THERE	A, C
ADIT	B
WE	C
GO	C

The keys of the hash table are the words, and the values tell you what pages each word appears on. Now suppose a user searches for *hi*. Let's see what pages *hi* shows up on.

Las claves de la tabla hash son las palabras y los valores le indican en qué páginas aparece cada palabra. Ahora supongamos que un usuario busca *hola*. Veamos en qué páginas aparece *hola*.

	HI	A, B
--	----	------

Aha! It appears on pages A and B. Let's show the user those pages as the result. Or suppose the user searches for *there*. Well, you know that it shows up on pages A and C. Pretty easy, huh? This is a useful data structure: a hash that maps words to places where they appear. This data structure is called an *inverted index*, and it's commonly used to build

search engines. If you're interested in search, this is a good place to start.

¡Ajá! Aparece en las páginas A y B. Mostremos al usuario esas páginas como resultado. O supongamos que el usuario busca allí. Bueno, sabes que aparece en las páginas A y C. Bastante fácil, ¿no? Esta es una estructura de datos útil: un hash que asigna palabras a los lugares donde aparecen. Esta estructura de datos se llama índice invertido y se usa comúnmente para crear motores de búsqueda. Si está interesado en la búsqueda, este es un buen lugar para comenzar.

## ***The Fourier transform***

### ***La transformada de Fourier***

The Fourier transform is one of those rare algorithms: brilliant, elegant, and with a million use cases. The best analogy for the Fourier transform comes from Better Explained (a great website that explains math simply): given a smoothie, the Fourier transform will tell you the ingredients in the smoothie.<sup>1</sup> Or, to put it another way, given a song, the transform can separate it into individual frequencies.

La transformada de Fourier es uno de esos raros algoritmos: brillante, elegante y con un millón de casos de uso. La mejor analogía para la transformada de Fourier proviene de BetterExplained (un fantástico sitio web que explica las

matemáticas de forma sencilla): dado un batido, la transformada de Fourier te dirá los ingredientes del batido.<sup>1</sup> O, para decirlo de otra manera, Dada una canción, la transformación puede separarla en frecuencias individuales.

It turns out that this simple idea has a lot of use cases. For example, if you can separate a song into frequencies, you can boost the ones you care about. You could boost the bass and hide the treble. The Fourier transform is great for processing signals. You can also use it to compress music. First, you break an audio file down into its ingredient notes. The Fourier transform tells you exactly how much each note contributes to the overall song. So you can get rid of the notes that aren't important. That's how the MP3 format works!

Resulta que esta sencilla idea tiene muchos casos de uso. Por ejemplo, si puedes separar una canción en frecuencias, puedes potenciar las que te interesan. Podrías potenciar los graves y ocultar los agudos. La transformada de Fourier es excelente para procesar señales. También puedes usarlo para comprimir música. Primero, divides un archivo de audio en sus notas de ingredientes. La transformada de Fourier te dice exactamente cuánto contribuye cada nota a la canción en general. Así podrás deshacerte de las notas que no son importantes. ¡Así funciona el formato MP3!

Music isn't the only type of digital signal. The JPG format is another compressed format, and it works the same way. People use the Fourier transform to try to predict upcoming earthquakes and analyze DNA. You can use it to build an app like Shazam, which guesses what song is playing. The

Fourier transform has a lot of uses. Chances are high that you'll run into it!

La música no es el único tipo de señal digital. El formato JPG es otro formato comprimido y funciona de la misma manera. La gente usa la transformada de Fourier para intentar predecir próximos terremotos y analizar el ADN. Puedes usarlo para crear una aplicación como Shazam, que adivina qué canción se está reproduciendo. La transformada de Fourier tiene muchos usos. ¡Hay muchas posibilidades de que te lo encuentres!

## ***Parallel algorithms***

### ***Algoritmos paralelos***

The next three topics are about scalability and working with a lot of data. Back in the day, computers kept getting faster and faster. If you wanted to make your algorithm faster, you could wait a few months, and the computers themselves would become faster. But we're near the end of that period. Instead, laptops and computers ship with multiple cores. To make your algorithms faster, you need to change them to run in parallel across all the cores at once!

Los siguientes tres temas tratan sobre la escalabilidad y el trabajo con una gran cantidad de datos. En el pasado, las computadoras eran cada vez más rápidas. Si quisieras hacer que tu algoritmo fuera más rápido, podrías esperar unos meses y las computadoras mismas serían más rápidas. Pero

estamos cerca del final de ese período. En cambio, las computadoras portátiles y de escritorio se entregan con múltiples núcleos. Para que sus algoritmos sean más rápidos, debe cambiarlos para que se ejecuten en paralelo en todos los núcleos a la vez.

Here's a simple example. The best you can do with a sorting algorithm is roughly  $O(n \log n)$ . It's well known that you can't sort an array in  $O(n)$  time—*unless you use a parallel algorithm!* There's a parallel version of quicksort that will sort an array in  $O(n)$  time.

He aquí un ejemplo sencillo. Lo mejor que puede hacer con un algoritmo de clasificación es aproximadamente  $O(n \log n)$ . Es bien sabido que no se puede ordenar una matriz en tiempo  $O(n)$ , ia menos que se utilice un algoritmo paralelo! Existe una versión paralela de ordenación rápida que ordenará una matriz en tiempo  $O(n)$ .

Parallel algorithms are hard to design. And it's hard to make sure they work correctly and to figure out what type of speed boost you'll see. One thing is for sure—the time gains aren't linear. So if you have two cores in your laptop instead of one, that almost never means your algorithm will magically run twice as fast. There are a couple of reasons for this:

Los algoritmos paralelos son difíciles de diseñar. Y es difícil asegurarse de que funcionen correctamente y determinar qué tipo de aumento de velocidad verá. Una cosa es segura: las ganancias de tiempo no son lineales. Entonces, si tiene dos núcleos en su computadora portátil en lugar de uno, eso

casi nunca significa que su algoritmo se ejecutará mágicamente dos veces más rápido. Hay un par de razones para esto:

- *Overhead of managing the parallelism*—Suppose you have to sort an array of 1,000 items. How do you divide this task between the two cores? Do you give each core 500 items to sort and then merge the two sorted arrays into one big sorted array? Merging the two arrays takes time.

Gastos generales de gestión del paralelismo:  
supongamos que tiene que ordenar una matriz de 1000 elementos. ¿Cómo se divide esta tarea entre los dos núcleos? ¿Le da a cada núcleo 500 elementos para ordenar y luego fusiona las dos matrices ordenadas en una gran matriz ordenada? Fusionar las dos matrices lleva tiempo.

- *Amdahl's law*—Suppose you are painting a picture. Paintings take you very long to do, typically 20 hours. Ideally, you would do it in 10 hours. You decide to optimize your process. You break it down into two steps: (i) doing the initial sketch and (ii) painting it. For the initial sketch, instead of free-handing it, surely tracing will be faster. But the next time you paint, it still takes 19 hours and 5 minutes! What happened? Well, the sketch used to take an hour. You've got it down to 5 minutes, which is a big improvement. But the painting is the part that took the most time, and you didn't optimize that at all. This is Amdahl's law. Amdahl's law says that when you optimize one part of the system, the performance gain is limited by how much time that part actually takes. In this case, we cut the sketch time to 1/12th of what it used to be. In the process, we saved 55 minutes. If we could have cut the painting time by the same amount, we would have saved 1,045 minutes! When you are speeding up your algorithm by parallelizing it, think about which part to parallelize. Are you parallelizing the painting part or the sketching part?

Ley de Amdahl: supongamos que estás pintando un cuadro. Las pinturas te llevan mucho tiempo, normalmente 20 horas. Lo ideal sería hacerlo en 10 horas. Tú decides optimizar tu proceso. Lo divides en dos pasos: (i) hacer el boceto inicial y (ii) pintarlo. Para el boceto inicial, en lugar de hacerlo a mano alzada, seguramente el calco será más rápido. Pero la próxima vez que pinges, itodavía te llevará 19 horas y 5 minutos! ¿Qué pasó? Bueno, el boceto solía tardar una hora. Lo has reducido a 5 minutos, lo cual es una gran mejora. Pero la pintura es la parte que tomó más tiempo y no la optimizaste en absoluto. Esta es la ley de Amdahl. La ley de Amdahl dice que cuando se optimiza una parte del sistema, la ganancia de rendimiento está limitada por el tiempo que realmente lleva esa parte. En este caso, reducimos el tiempo del boceto a  $1/12$  de lo que solía ser. En el proceso ahorrados 55 minutos. Si hubiéramos podido reducir el tiempo de pintura en la misma cantidad, ihabríamos ahorrado 1.045 minutos! Cuando esté acelerando su algoritmo al paralelizarlo, piense en qué parte paralelizar. ¿Estás paralelizando la parte de pintar o la de dibujar?

- *Load balancing*—Suppose you have 10 tasks to do, so you give each core five tasks. But core A gets all the easy tasks, so it's done in 10 seconds, whereas core B gets all the hard tasks, so it takes a minute. That means core A was sitting idle for 50 seconds while core B was doing all the work! How do you distribute the work evenly so both cores are working equally hard?

Equilibrio de carga: supongamos que tiene 10 tareas que realizar, por lo que le asigna cinco tareas principales a cada una. Pero el núcleo A realiza todas las tareas fáciles, por lo que se realiza en 10 segundos, mientras que el núcleo B realiza todas las tareas difíciles, por lo que lleva un minuto. ¡Eso significa que el núcleo A estuvo inactivo durante 50 segundos mientras el núcleo B hacía todo el trabajo! ¿Cómo se distribuye el trabajo de manera uniforme para que ambos núcleos trabajen con la misma intensidad?

If you're interested in the theoretical side of performance and scalability, parallel algorithms might be for you!

Si está interesado en el aspecto teórico del rendimiento y la escalabilidad, los algoritmos paralelos pueden ser para usted!

## **map/reduce**

### **Mapa reducido**

There's a special type of parallel algorithm that is becoming increasingly popular: the *distributed algorithm*. It's fine to run a parallel algorithm on your laptop if you need two to four cores, but what if you need hundreds of cores? Then you can write your algorithm to run across multiple machines. Google popularized a distributed algorithm that they called MapReduce, but these functions have been around for longer.

Existe un tipo especial de algoritmo paralelo que se está volviendo cada vez más popular: el algoritmo distribuido. Está bien ejecutar un algoritmo paralelo en su computadora portátil si necesita de dos a cuatro núcleos, pero ¿qué pasa si necesita cientos de núcleos? Luego puede escribir su algoritmo para que se ejecute en varias máquinas. Google popularizó un algoritmo distribuido al que llamaron MapReduce, pero estas funciones existen desde hace más tiempo.

Why are distributed algorithms useful? Suppose you have a table with billions or trillions of rows, and you want to run a complicated SQL query on it. You can't run it on MySQL because it struggles after a few billion rows. Use map/reduce!

¿Por qué son útiles los algoritmos distribuidos? Supongamos que tiene una tabla con miles de millones o billones de filas

y desea ejecutar una consulta SQL complicada en ella. No puedes ejecutarlo en MySQL porque tiene problemas después de unos miles de millones de filas. ¡Usa mapa/reduce!

Or suppose you have to process a long list of jobs. Each job takes 10 seconds to process, and you need to process 1 million jobs like this. If you do this on one machine, it will take you months! If you could run it across 100 machines, you might be done in a few days.

O supongamos que tiene que procesar una larga lista de trabajos. Cada trabajo tarda 10 segundos en procesarse y usted necesita procesar 1 millón de trabajos como este. Si haces esto en una máquina, te llevará meses! Si pudiera ejecutarlo en 100 máquinas, podría terminar en unos días.

Distributed algorithms are great when you have a lot of work to do and want to speed up the time required to do it.

Los algoritmos distribuidos son excelentes cuando tienes mucho trabajo que hacer y quieres acelerar el tiempo necesario para hacerlo.

## ***Bloom filters and HyperLogLog***

### ***Filtros Bloom y HyperLogLog***

Suppose you're running Reddit. When someone posts a link, you want to see if it's been posted before. Stories that haven't been posted before are considered more valuable.

So you need to figure out whether this link has been posted before.

Suponga que está ejecutando Reddit. Cuando alguien publica un enlace, querrás ver si ya se ha publicado antes. Las historias que no se han publicado antes se consideran más valiosas. Por lo tanto, debe averiguar si este enlace se publicó anteriormente.

Or suppose you're Google, and you're crawling web pages. You only want to crawl a web page if you haven't crawled it already. So you need to figure out whether this page has been crawled before.

O supongamos que eres Google y estás rastreando páginas web. Sólo desea rastrear una página web si aún no la ha rastreado. Por lo tanto, debes averiguar si esta página se ha rastreado antes.

Or suppose you're running bit.ly, which is a URL shortener. You don't want to redirect users to malicious websites. You have a set of URLs that are considered malicious. Now you need to figure out whether you're redirecting the user to a URL in that set.

O suponga que está ejecutando bit.ly, que es un acortador de URL. No desea redirigir a los usuarios a sitios web maliciosos. Tiene un conjunto de URL que se consideran maliciosas. Ahora necesitas determinar si estás redirigiendo al usuario a una URL de ese conjunto.

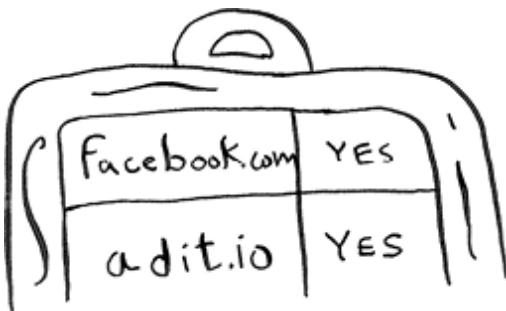
All of these examples have the same problem. You have a very large set.

Todos estos ejemplos tienen el mismo problema. Tienes un conjunto muy grande.



Now you have a new item, and you want to see whether it belongs in that set. You could do this quickly with a hash. For example, suppose Google has a big hash in which the keys are all the pages it has crawled.

Ahora tiene un elemento nuevo y desea ver si pertenece a ese conjunto. Podrías hacer esto rápidamente con un hash. Por ejemplo, supongamos que Google tiene un hash grande en el que las claves son todas las páginas que ha rastreado.



You want to see whether you've already crawled adit.io. Look it up in the hash.

Quieres ver si ya has rastreado adit.io. Búscalos en el hash.

*adit.io → YES*

adit.io is a key in the hash, so you've already crawled it. The average lookup time for hash tables is O(1). adit.io is in the hash, so you've already crawled it. You found that out in constant time. Pretty good!

adit.io es una clave en el hash, por lo que ya la has rastreado. El tiempo medio de búsqueda de tablas hash es O(1). adit.io está en el hash, por lo que ya lo has rastreado. Lo descubriste en tiempo constante. ¡Bastante bien!

Except that this hash needs to be *huge*. Google indexes trillions of web pages. If this hash has all the URLs that Google has indexed, it will take up a lot of space. Reddit and bit.ly have the same space problem. When you have so much data, you need to get creative!

Excepto que este hash debe ser enorme. Google indexa billones de páginas web. Si este hash tiene todas las URL que Google ha indexado, ocupará mucho espacio. Reddit y bit.ly tienen el mismo problema de espacio. Cuando tienes tantos datos, necesitas ser creativo!

## Bloom filters

### Filtros de floración

Bloom filters offer a solution. Bloom filters are *probabilistic data structures*. They give you an answer that could be wrong but is probably correct. Instead of a hash, you can ask your bloom filter if you've crawled this URL before. A hash table would give you an accurate answer. A bloom filter will give you an answer that's probably correct:

Los filtros Bloom ofrecen una solución. Los filtros Bloom son estructuras de datos probabilísticos. Te dan una respuesta que podría estar equivocada pero probablemente sea correcta. En lugar de un hash, puedes preguntarle a tu filtro de floración si has rastreado esta URL antes. Una tabla hash le daría una respuesta precisa. Un filtro de floración le dará una respuesta que probablemente sea correcta:

- False positives are possible. Google might say, "You've already crawled this site," even though you haven't.
- Los falsos positivos son posibles. Google podría decir: "Ya has rastreado este sitio", aunque no lo hayas hecho.

- False negatives aren't possible. If the bloom filter says, "You haven't crawled this site," then you *definitely* haven't crawled this site.

Los falsos negativos no son posibles. Si el filtro de floración dice: "No has rastreado este sitio", entonces definitivamente no has rastreado este sitio.

Bloom filters are great because they take up very little space. A hash table would have to store every URL crawled by Google, but a bloom filter doesn't have to do that. They're great when you don't need an exact answer, as in all of these examples. It's OK for bit.ly to say, "We think this site might be malicious, so be extra careful."

Los filtros Bloom son geniales porque ocupan muy poco espacio. Una tabla hash tendría que almacenar cada URL rastreada por Google, pero un filtro de floración no tiene por qué hacerlo. Son geniales cuando no necesitas una respuesta exacta, como en todos estos ejemplos. Está bien que bit.ly diga: "Creemos que este sitio podría ser malicioso, así que tenga mucho cuidado".

## HyperLogLog

### HyperLogLog

Along the same lines is another algorithm called HyperLogLog. Suppose Google wants to count the number of *unique* searches performed by its users. Or suppose Amazon wants to count the number of unique items that users looked at today. Answering these questions takes a lot of space!

With Google, you'd have to keep a log of all the unique searches. When a user searches for something, you have to see whether it's already in the log. If not, you have to add it to the log. Even for a single day, this log would be massive!

En la misma línea hay otro algoritmo llamado HyperLogLog. Supongamos que Google quiere contar el número de búsquedas únicas realizadas por sus usuarios. O supongamos que Amazon quiere contar la cantidad de artículos únicos que los usuarios miraron hoy. ¡Responder estas preguntas requiere mucho espacio! Con Google, tendrías que mantener un registro de todas las búsquedas únicas. Cuando un usuario busca algo, hay que ver si ya está en el registro. Si no, debes agregarlo al registro. ¡Incluso por un solo día, este registro sería enorme!

HyperLogLog approximates the number of unique elements in a set. Just like bloom filters, it won't give you an exact answer, but it comes very close and uses only a fraction of the memory a task like this would otherwise take.

HyperLogLog se aproxima al número de elementos únicos en un conjunto. Al igual que los filtros de floración, no le dará una respuesta exacta, pero se acerca mucho y utiliza sólo una fracción de la memoria que de otro modo requeriría una tarea como esta.

If you have a lot of data and are satisfied with approximate answers, check out probabilistic algorithms!

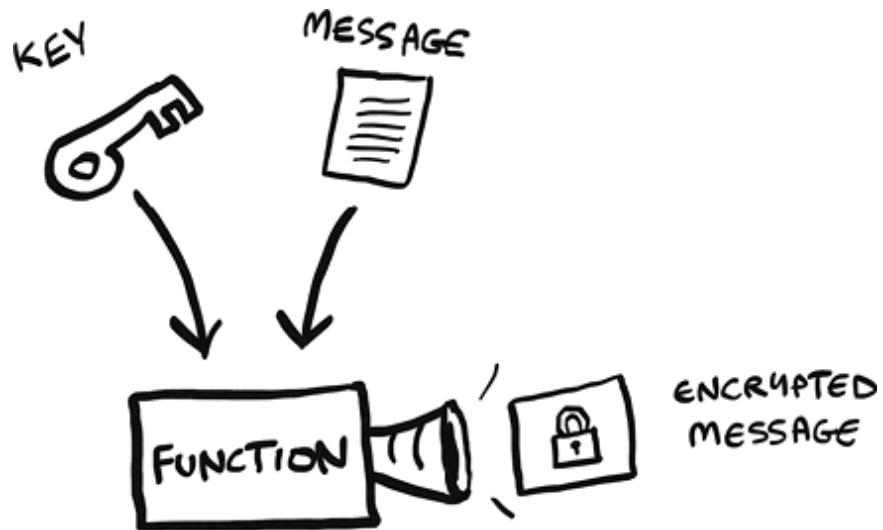
Si tiene muchos datos y está satisfecho con las respuestas aproximadas, iconsulte los algoritmos probabilísticos!

## ***HTTPS and the Diffie–Hellman key exchange***

### ***HTTPS y el intercambio de claves Diffie–Hellman***

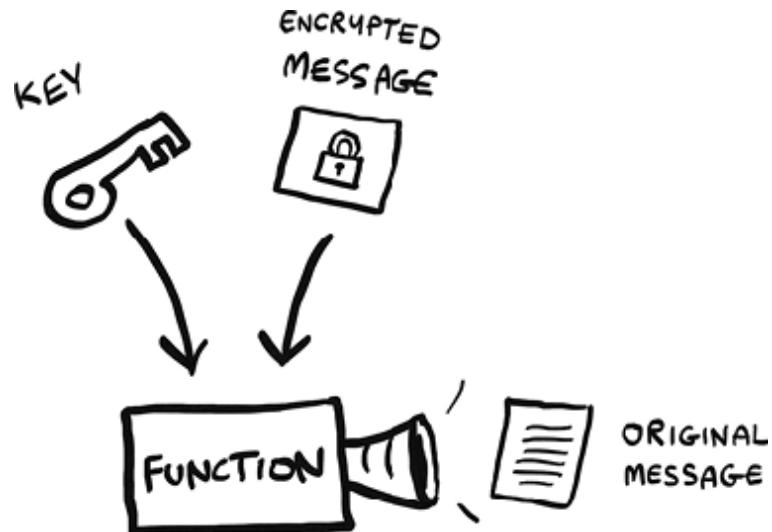
HTTPS is the backbone of the internet, enabling secure online transactions from password entry to online purchases. HTTPS works by encrypting messages between the client and the server. Here's how encryption works. You pass a message and a secret key into a function. It then generates an encrypted message.

HTTPS es la columna vertebral de Internet y permite transacciones seguras en línea, desde el ingreso de contraseñas hasta las compras en línea. HTTPS funciona cifrando mensajes entre el cliente y el servidor. Así es como funciona el cifrado. Pasas un mensaje y una clave secreta a una función. Luego genera un mensaje cifrado.



To decrypt the message, pass the encrypted message and the *same key* into a function, and you will get back the original message.

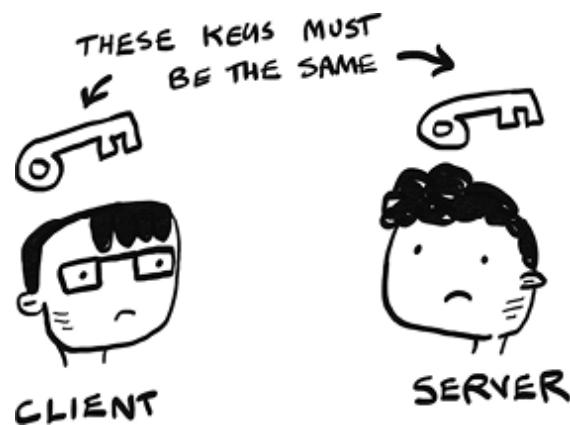
Para descifrar el mensaje, pase el mensaje cifrado y la misma clave a una función y recibirá el mensaje original.



When you send some data to a server, your browser encrypts the message for you, and then the server decrypts

it. Simple, right? Except for one thing: How do you make sure your browser and the server have the same key?

Cuando envía algunos datos a un servidor, su navegador cifra el mensaje por usted y luego el servidor lo descifra. Sencillo, ¿verdad? Excepto por una cosa: ¿Cómo se asegura de que su navegador y el servidor tengan la misma clave?



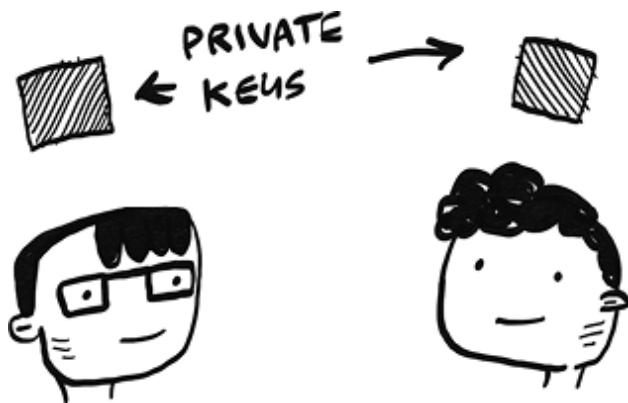
Remember that for HTTPS to work, both sides need to have the same key. But how do you agree on a key without someone seeing what it is? If you send the server a key to use, someone could intercept that key. How do you agree on a key so that only your browser and the server know what key you're using? This seems impossible, but it can be done! There's a very clever algorithm to do it called the Diffie-Hellman key exchange. Here's how it works.

Recuerde que para que HTTPS funcione, ambas partes deben tener la misma clave. Pero, ¿cómo se pone de acuerdo sobre una clave sin que nadie vea cuál es? Si envía al servidor una clave para usar, alguien podría interceptar esa clave. ¿Cómo se acuerda una clave para que solo su navegador y el

servidor sepan qué clave está utilizando? ¡Esto parece imposible, pero se puede hacer! Existe un algoritmo muy inteligente para hacerlo llamado intercambio de claves Diffie-Hellman. Así es como funciona.

In step 1, we generate our own keys. I'm the client, and I generate a key for myself. The server also generates a key. These keys are different. We don't know each other's keys. They are private to us.

En el paso 1, generamos nuestras propias claves. Soy el cliente y genero una clave para mí. El servidor también genera una clave. Estas claves son diferentes. No conocemos las claves de cada uno. Son privados para nosotros.

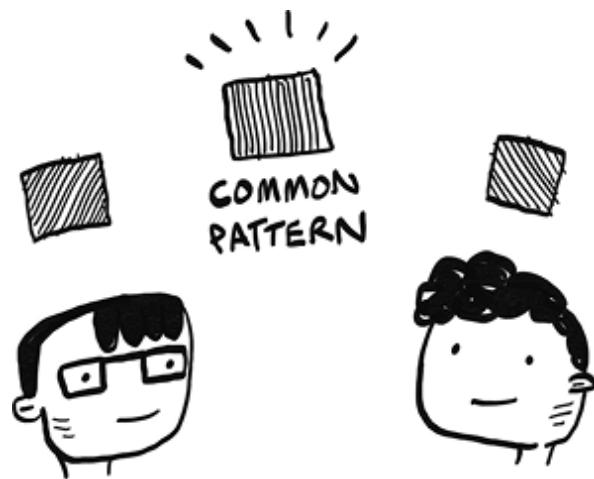


I'm using a pattern here for each key so I can show you visually what happens. In reality, these would be bytes.

Estoy usando un patrón aquí para cada tecla para poder mostrarte visualmente lo que sucede. En realidad, serían bytes.

In step 2, we generate a common pattern.

En el paso 2, generamos un patrón común.

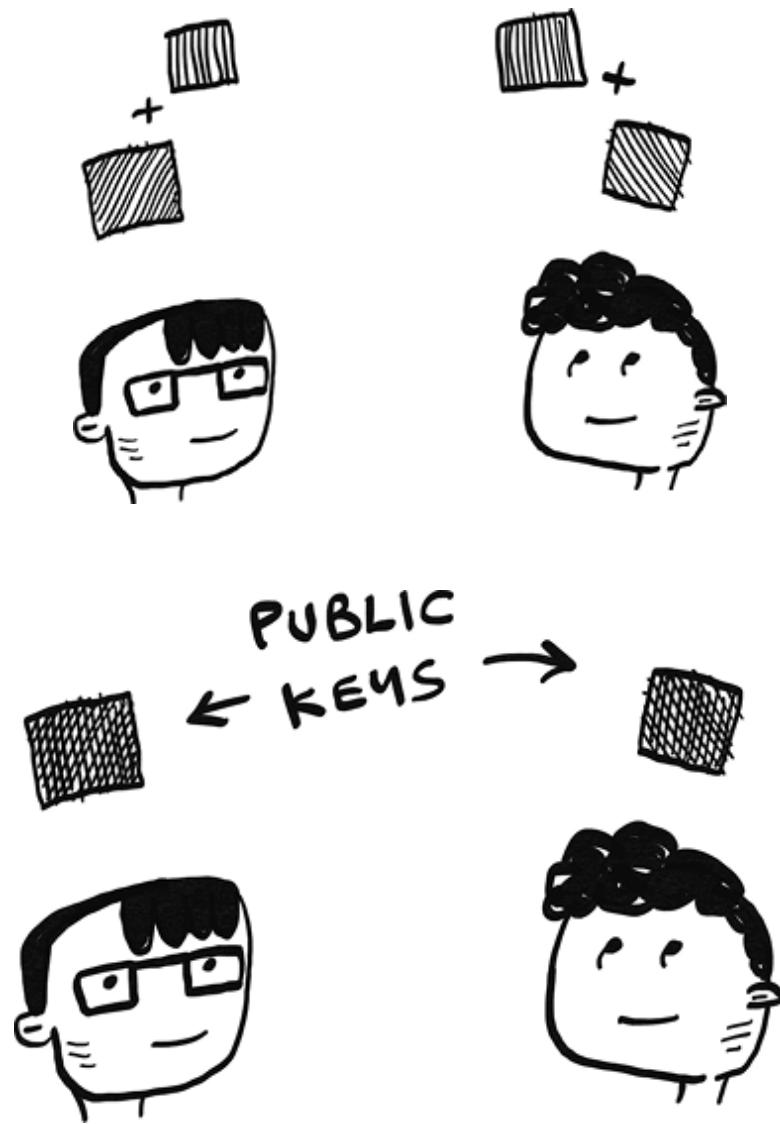


This pattern is public. Both of us can see it as well as anyone else. We don't care who sees it.

Este patrón es público. Ambos podemos verlo tan bien como cualquier otra persona. No nos importa quién lo vea.

In step 3, we each overlay this pattern onto our private key.

En el paso 3, cada uno de nosotros superponemos este patrón en nuestra clave privada.

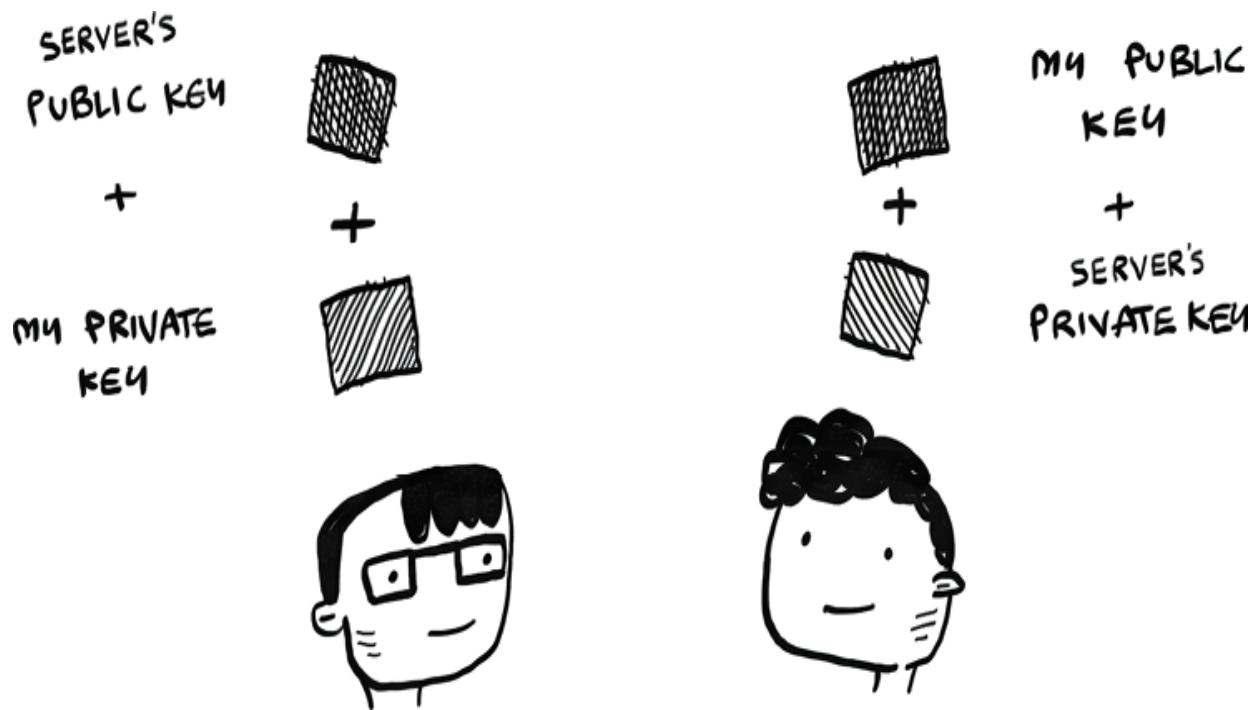


Doing so gives us our *public key*. The public key is, well, public, so we don't care who sees it. The server can see my public key, and I can see its public key.

Hacerlo nos da nuestra clave pública. La clave pública es, bueno, pública, por lo que no nos importa quién la vea. El servidor puede ver mi clave pública y yo puedo ver su clave pública.

Finally, in step 4, I take the server's public key and overlay it onto my private key. And the server does the same with my public key.

Finalmente, en el paso 4, tomo la clave pública del servidor y la superpongo a mi clave privada. Y el servidor hace lo mismo con mi clave pública.



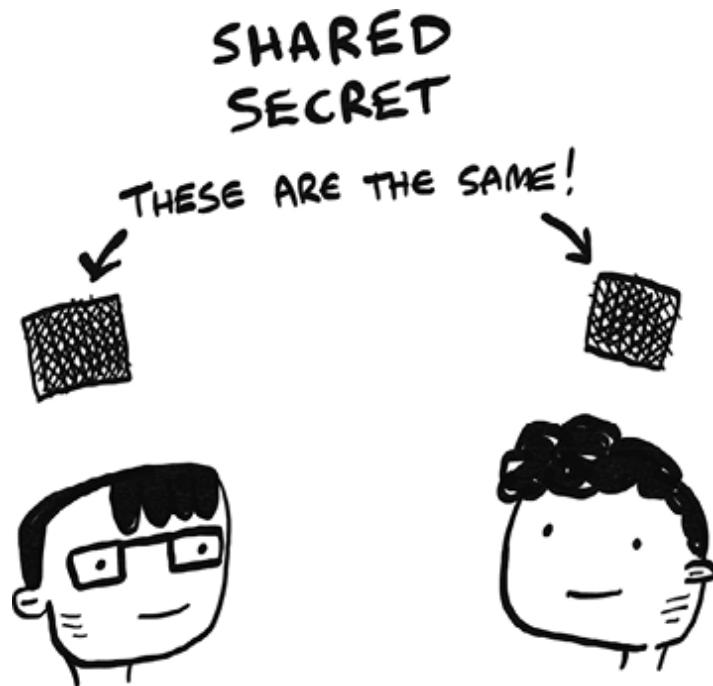
Tada! We now have the same key! We both have a key that combines three patterns.

¡Tada! ¡Ahora tenemos la misma clave! Ambos tenemos una clave que combina tres patrones.



Somehow we have both agreed on a key without ever sending the key to each other. This key we have agreed on is called the *shared secret*, and that's how the Diffie-Hellman key exchange works.

De alguna manera, ambos hemos acordado una clave sin ni siquiera enviárnosla el uno al otro. Esta clave que hemos acordado se llama secreto compartido y así es como funciona el intercambio de claves Diffie-Hellman.



HTTPS is a fascinating and important part of our daily lives. Here are some terms you'll hear about in connection to HTTPS:

HTTPS es una parte fascinante e importante de nuestra vida diaria. Aquí hay algunos términos que escuchará en relación con HTTPS:

- *TLS*—TLS (Transport Layer Security) is a protocol. TLS is how we establish this secure connection.  
TLS: TLS (Seguridad de la capa de transporte) es un protocolo. TLS es la forma en que establecemos esta conexión segura.
- *SSL*—SSL is the old name for TLS, but people often don't make a distinction. If you hear someone say SSL, they are probably talking about TLS. People find security holes in these protocols, so they constantly need to be updated. The TLS protocol was first introduced in 1999. Every version of the SSL protocol that came before the TLS protocol is broken.

SSL: SSL es el nombre antiguo de TLS, pero la gente suele no hacer distinción. Si escucha a alguien decir SSL, probablemente esté hablando de TLS. La gente encuentra agujeros de seguridad en estos protocolos, por lo que es necesario actualizarlos constantemente. El protocolo TLS se introdujo por primera vez en 1999. Todas las versiones del protocolo SSL anteriores al protocolo TLS no funcionan.

- *Symmetric key encryption*—In our example, both sides used the same key. There is also something called *asymmetric key encryption*, where both sides have different keys. I talked about symmetric key encryption because that is what HTTPS uses.

Cifrado de clave simétrica: en nuestro ejemplo, ambas partes utilizaron la misma clave. También existe algo llamado cifrado de clave asimétrica, donde ambas partes tienen claves diferentes. Hablé de cifrado de clave simétrica porque eso es lo que utiliza HTTPS.

HTTPS uses a modified version of the Diffie–Hellman key exchange called the *ephemeral Diffie–Hellman key exchange*. It works exactly like we just saw, except the private keys are generated fresh for every connection. This means even if an attacker discovers one of the private keys, they can only decrypt the messages from one connection.

HTTPS utiliza una versión modificada del intercambio de claves Diffie–Hellman llamado intercambio de claves efímero Diffie–Hellman. Funciona exactamente como acabamos de ver, excepto que las claves privadas se generan nuevas para cada conexión. Esto significa que incluso si un atacante descubre una de las claves privadas, solo podrá descifrar los mensajes desde una conexión.

The world of cryptography is deep and interesting. If you'd like to learn more, I highly recommend another Manning book: *Real-World Cryptography* by David Wong (<https://www.manning.com/books/real-world-cryptography>).

El mundo de la criptografía es profundo e interesante. Si desea obtener más información, le recomiendo otro libro de Manning: Real-World Cryptography de David Wong (<https://www.manning.com/books/real-world-cryptography>).



## ***Locality-sensitive hashing***

### ***Hashing sensible a la localidad***

A lot of hash functions you use will be locality insensitive. Suppose you have a string, and you generate a hash for it using SHA-256.

Muchas de las funciones hash que utilice no tendrán en cuenta la localidad. Suponga que tiene una cadena y genera un hash para ella utilizando SHA-256.

*dog → cd6357*

If you change just one character of the string and regenerate the hash, it's totally different!

Si cambias solo un carácter de la cadena y regeneras el hash, es totalmente diferente!

dot → e392da

This is good because an attacker can't compare hashes to see whether they're close to cracking a password.

Esto es bueno porque un atacante no puede comparar hashes para ver si está cerca de descifrar una contraseña.

Sometimes, you want the opposite: you want a locality-sensitive hash function. That's where *Simhash* comes in. If you make a small change to a string, Simhash generates a hash that's only a little different. This allows you to compare hashes and see how similar two strings are, which is pretty useful!

A veces, desea lo contrario: desea una función hash sensible a la localidad. Ahí es donde entra Simhash. Si realiza un pequeño cambio en una cadena, Simhash genera un hash que es sólo un poco diferente. Esto le permite comparar hashes y ver qué tan similares son dos cadenas, lo cual es bastante útil!

- Google uses Simhash to detect duplicates while crawling the web.

Google utiliza Simhash para detectar duplicados mientras rastrea la web.

- A teacher could use Simhash to see whether a student copied an essay from the web.

Un profesor podría utilizar Simhash para ver si un alumno copió un ensayo de la web.

- Scribd allows users to upload documents or books to share with others. But Scribd doesn't want users uploading copyrighted content! The site could use Simhash to check whether an upload is similar to a *Harry Potter* book and, if so, reject it automatically.

Scribd permite a los usuarios cargar documentos o libros para compartir con otros. ¡Pero Scribd no quiere que los usuarios carguen contenido protegido por derechos de autor! El sitio podría utilizar Simhash para comprobar si una carga es similar a un libro de Harry Potter y, de ser así, rechazarla automáticamente.

Simhash is useful when you want to check for similar items.

Simhash es útil cuando desea buscar elementos similares.

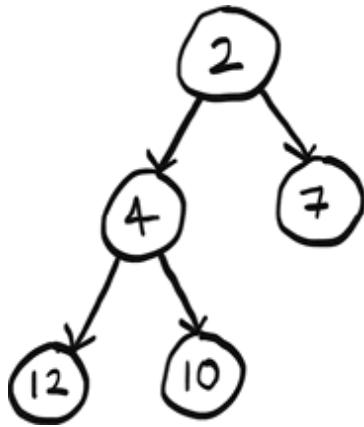
## ***Min heaps and priority queues***

### ***Montones mínimos y colas prioritarias***

Min heaps are a data structure built using trees. These heaps store a bunch of numbers. Here is an example of a min heap.

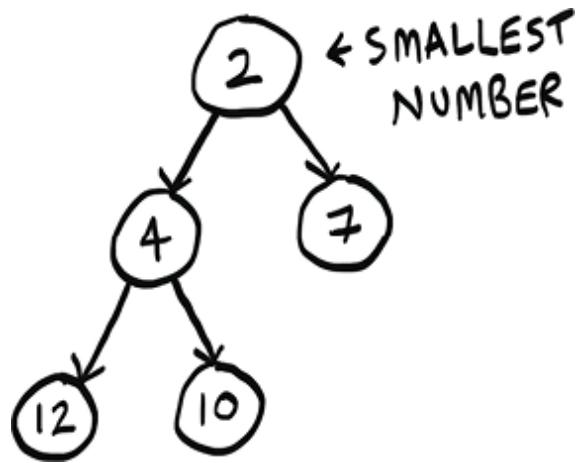
Los montones mínimos son una estructura de datos construida utilizando árboles. Estos montones almacenan

una gran cantidad de números. A continuación se muestra un ejemplo de un montón mínimo.



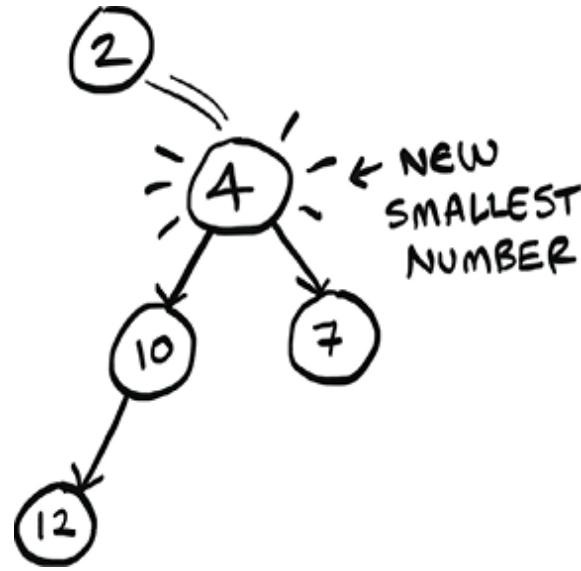
Min heaps let you get the smallest element in the heap quickly since the smallest value is always the root. This is the main usefulness of the min heap. You can look at the smallest element in O(1) time.

Los montones mínimos le permiten obtener el elemento más pequeño del montón rápidamente, ya que el valor más pequeño es siempre la raíz. Ésta es la principal utilidad del montón mínimo. Puedes mirar el elemento más pequeño en tiempo O(1).



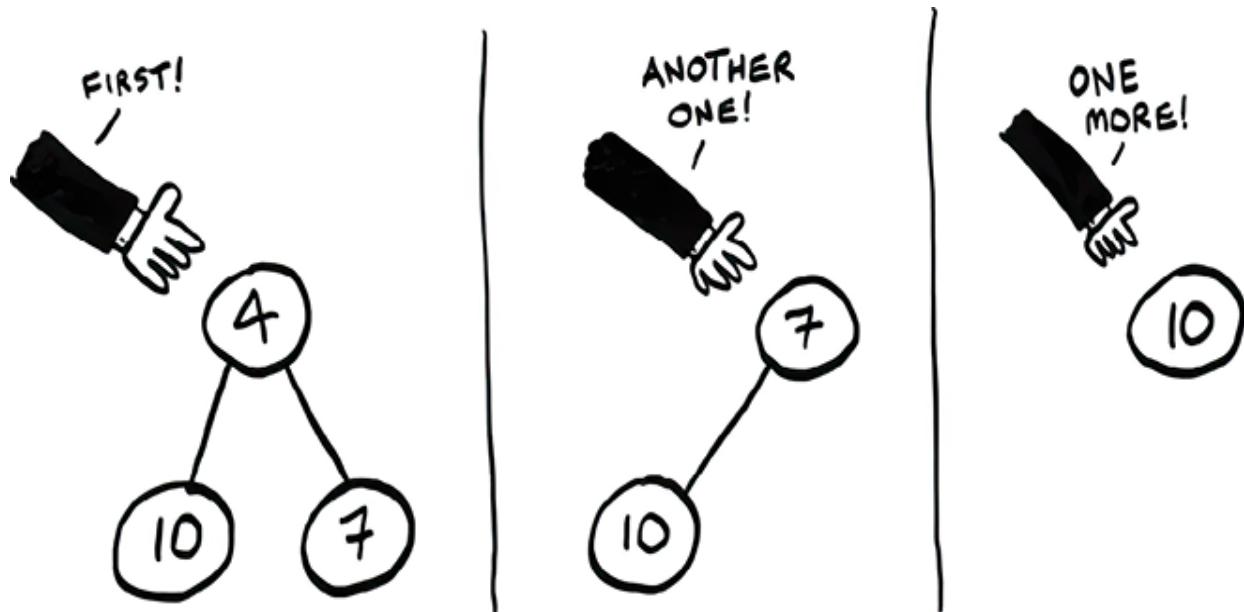
Or, in  $O(\log n)$  time, you can remove it from the heap with a new min in its place:

O, en tiempo  $O(\log n)$ , puedes eliminarlo del montón con un nuevo mínimo en su lugar:



Heaps let you sort very easily. Keep asking for the minimum value.

Los montones te permiten ordenar muy fácilmente. Sigue preguntando por el valor mínimo.

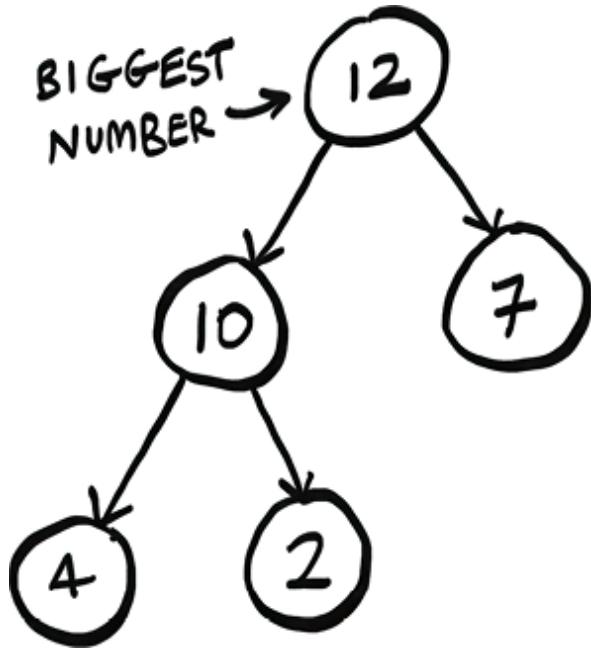


And keep putting the values in order. At the end, the tree will be empty, and you will have a sorted list of numbers! This algorithm is called *heapsort*.

Y sigue poniendo los valores en orden. ¡Al final, el árbol estará vacío y tendrás una lista ordenada de números! Este algoritmo se llama *heapsort*.

Max heaps are very similar to min heaps, but now the root is the largest value.

Los montones máximos son muy similares a los montones mínimos, pero ahora la raíz es el valor más grande.



Heaps are great for implementing priority queues. We met the queue data structure in chapter 6. Remember that queues are a FIFO (first in, first out) data structure. (In contrast, a stack is a LIFO [last in, first out] data structure.) Well, a priority queue is just like a queue, except when you ask for an item, it gives you the item with the highest priority! A to-do list application is a great use case for a priority queue. First, you put in your to-dos. Then you ask for something to work on, and your priority queue gives you the next highest priority to-do. Priority queues are also used to implement an efficient version of Dijkstra's algorithm.

Los montones son excelentes para implementar colas prioritarias. Conocimos la estructura de datos de la cola en el capítulo 6. Recuerde que las colas son una estructura de datos FIFO (primero en entrar, primero en salir). (Por el contrario, una pila es una estructura de datos LIFO [último en entrar, primero en salir].) Bueno, una cola de prioridad es

como una cola, excepto que cuando pides un artículo, te da el artículo con mayor prioridad. Una aplicación de lista de tareas pendientes es un excelente caso de uso para una cola de prioridad. Primero, pones tus tareas pendientes. Luego pides algo en lo que trabajar y tu cola de prioridades te da la siguiente prioridad más alta. Las colas de prioridad también se utilizan para implementar una versión eficiente del algoritmo de Dijkstra.

## ***Linear programming***

## ***Programación lineal***

I saved the best for last. Linear programming is one of the coolest things I know.

Guardé lo mejor para el final. La programación lineal es una de las cosas más interesantes que conozco.

Linear programming is used to maximize something given some linear constraints. For example, suppose your company makes two products, shirts and totes. Shirts need 1 m of fabric and five buttons. Totes need 2 m of fabric and two buttons. You have 11 meters of fabric and 20 buttons. You make \$2 per shirt and \$3 per tote. How many shirts and totes should you make to maximize your profit?

La programación lineal se utiliza para maximizar algo dadas algunas restricciones lineales. Por ejemplo, supongamos que su empresa fabrica dos productos, camisas y bolsos. Las camisas necesitan 1 m de tela y cinco botones. Los bolsos

necesitan 2 m de tela y dos botones. Tienes 11 metros de tela y 20 botones. Ganas \$2 por camisa y \$3 por bolso. ¿Cuántas camisas y bolsos deberías fabricar para maximizar tus ganancias?

Here you're trying to maximize profit, and you're constrained by the amount of materials you have.

Aquí estás tratando de maximizar las ganancias y estás limitado por la cantidad de materiales que tienes.

Another example: you're a politician, and you want to maximize the number of votes you get. Your research has shown that it takes an average of 1 hour of work (marketing, research, and so on) for each vote from a San Franciscan or 1.5 hours/vote from a Chicagoan. You need at least 500 San Franciscans and at least 300 Chicagoans. You have 50 days. It costs you \$2/San Franciscan versus \$1/Chicagoan. Your total budget is \$1,500. What's the maximum number of total votes you can get (San Francisco + Chicago)?

Otro ejemplo: eres un político y quieres maximizar el número de votos que obtienes. Su investigación ha demostrado que se necesita un promedio de 1 hora de trabajo (marketing, investigación, etc.) por cada voto de un ciudadano de San Francisco o 1,5 horas/voto de un ciudadano de Chicago. Se necesitan al menos 500 habitantes de San Francisco y al menos 300 habitantes de Chicago. Tienes 50 días. Le cuesta \$2/San Franciscano versus \$1/Chicagoano. Su presupuesto total es de \$1,500.

¿Cuál es el número máximo de votos totales que puede obtener (San Francisco + Chicago)?

Here you're trying to maximize votes, and you're constrained by time and money.

Aquí estás intentando maximizar los votos y estás limitado por el tiempo y el dinero.

You might be thinking, "You've talked about a lot of optimization topics in this book. How are they related to linear programming?" All the graph algorithms we have discussed in this book can be done through linear programming instead. Linear programming is a much more general framework, and the graph problems we have seen are a subset of that. I hope your mind is blown!

Quizás esté pensando: "Has hablado de muchos temas de optimización en este libro. ¿Cómo se relacionan con la programación lineal? Todos los algoritmos gráficos que hemos analizado en este libro se pueden realizar mediante programación lineal. La programación lineal es un marco mucho más general y los problemas de gráficos que hemos visto son un subconjunto de él. ¡Espero que te hayas vuelto loco!"

Linear programming uses the Simplex algorithm. It's a complex algorithm, which is why I didn't include it in this book. If you're interested in optimization, look up linear programming!

La programación lineal utiliza el algoritmo Simplex. Es un algoritmo complejo, por eso no lo incluí en este libro. Si está interesado en la optimización, ¡busque programación lineal!

## ***Epilogue***

## ***Epílogo***

I hope this quick tour of 10 algorithms showed you how much more is left to discover. I think the best way to learn is to find something you're interested in and dive in. This book gives you a solid foundation to do just that.

Espero que este recorrido rápido por 10 algoritmos le haya mostrado cuánto más queda por descubrir. Creo que la mejor manera de aprender es encontrar algo que le interese y sumergirse en ello. Este libro le brinda una base sólida para hacer precisamente eso.

---

<sup>1</sup> Kalid, “An Interactive Guide to the Fourier Transform,” Better Explained,  
<http://mng.bz/dd9N>.

<sup>1</sup> Kalid, “Una guía interactiva para la transformada de Fourier”, mejor explicada,  
<http://mng.bz/dd9N>.

## ***Appendix A. Performance of AVL trees***

---

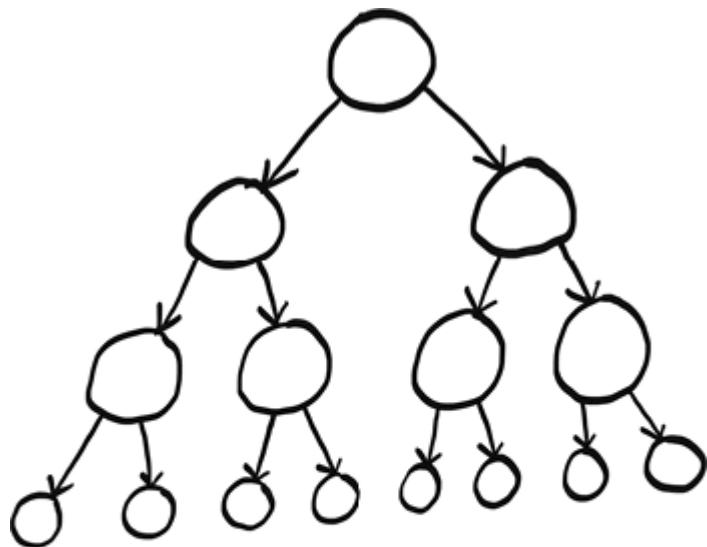
### ***Apéndice A. Rendimiento de los árboles***

#### ***AVL***

---

This appendix discusses the performance of AVL trees, which were introduced in chapter 8. You will need to read that chapter before reading this.

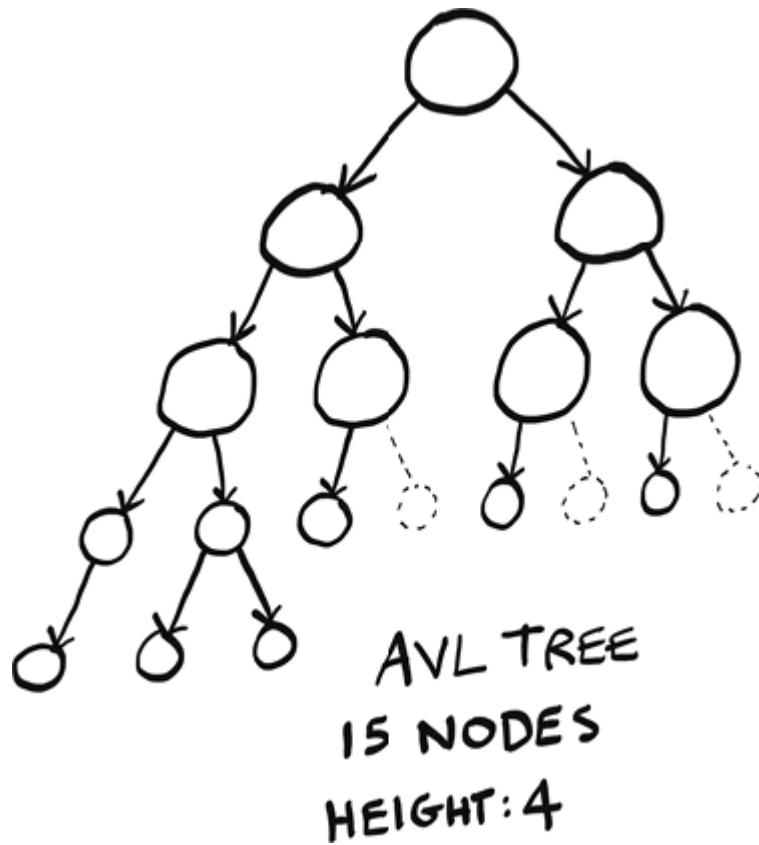
Este apéndice analiza el rendimiento de los árboles AVL, que se presentaron en el capítulo 8. Deberá leer ese capítulo antes de leer esto.



PERFECTLY BALANCED TREE  
15 NODES  
HEIGHT: 3

Remember that AVL trees offer  $O(\log n)$  search performance. But there is something misleading going on. Here are two trees. Both offer  $O(\log n)$  search performance, but their heights are different!

Recuerde que los árboles AVL ofrecen un rendimiento de búsqueda  $O(\log n)$ . Pero algo engañoso está sucediendo. Aquí hay dos árboles. Ambos ofrecen un rendimiento de búsqueda  $O(\log n)$ , ipero sus alturas son diferentes!



(Dashed nodes show the holes in the tree.)

(Los nodos discontinuos muestran los agujeros en el árbol).

AVL trees allow a difference of one in heights. That's why, even though both these trees have 15 nodes, the perfectly balanced tree is height 3, but the AVL tree is height 4. The perfectly balanced tree is what we might picture a balanced tree to look like, where each level is completely filled with nodes before a new level is added. But the AVL tree is also considered "balanced," even though it has holes—gaps where a node could be.

Los árboles AVL permiten una diferencia de uno en alturas. Es por eso que, aunque ambos árboles tienen 15 nodos, el

árbol perfectamente equilibrado tiene una altura de 3, pero el árbol AVL tiene una altura de 4. El árbol perfectamente equilibrado es lo que podríamos imaginar como sería un árbol equilibrado, donde cada nivel está completamente lleno, con nodos antes de agregar un nuevo nivel. Pero el árbol AVL también se considera "equilibrado", aunque tiene agujeros (espacios donde podría haber un nodo).

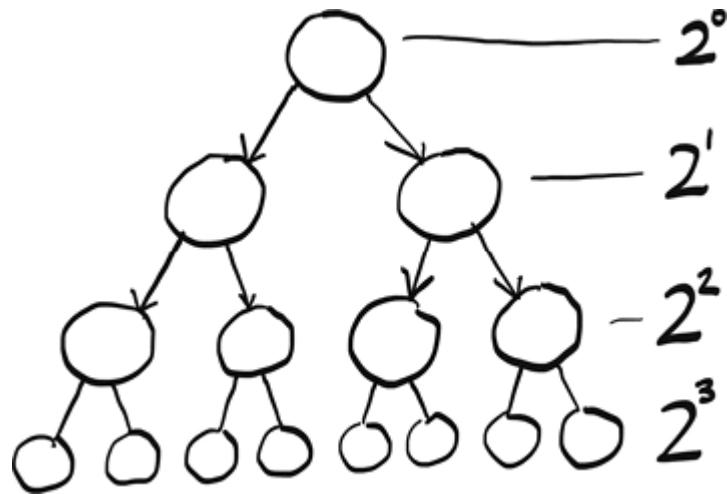
Remember that in a tree, performance is closely related to height. How can these trees offer the same performance if their heights are different? Well, we never discussed what the base in  $\log n$  is!

Recuerde que en un árbol el rendimiento está muy relacionado con la altura. ¿Cómo pueden estos árboles ofrecer el mismo rendimiento si sus alturas son diferentes? Bueno, inunca discutimos cuál es la base en  $\log n$ !

The perfectly balanced tree has performance  $O(\log n)$ , where the “log” is log base 2, just like binary search. We can see that in the picture. Each new level doubles the nodes plus 1. So a perfectly balanced tree of height 1 has 3 nodes, of height 2 has 7 nodes ( $3 \cdot 2 + 1$ ), of height 3 has 15 nodes ( $7 \cdot 2 + 1$ ), etc. You could also think of it as each layer adds a number of nodes equal to a power of 2.

El árbol perfectamente equilibrado tiene un rendimiento  $O(\log n)$ , donde el “log” es log base 2, al igual que la búsqueda binaria. Podemos ver eso en la imagen. Cada nuevo nivel duplica los nodos más 1. Entonces, un árbol perfectamente equilibrado de altura 1 tiene 3 nodos, de

altura 2 tiene 7 nodos ( $32 + 1$ ), de altura 3 tiene 15 nodos ( $72 + 1$ ), etc. Piense en ello como si cada capa agregara una cantidad de nodos igual a una potencia de 2.



So the perfectly balanced tree has performance  $O(\log n)$ , where the "log" is log base 2.

Entonces, el árbol perfectamente equilibrado tiene un rendimiento  $O(\log n)$ , donde el "log" es log en base 2.

The AVL tree has some gaps. In an AVL tree, each new layer adds less than double the nodes. It turns out that an AVL offers performance  $O(\log n)$ , but the "log" is log base *phi* (aka the golden ratio, aka  $\sim 1.618$ ).

El árbol AVL tiene algunas lagunas. En un árbol AVL, cada nueva capa agrega menos del doble de nodos. Resulta que un AVL ofrece un rendimiento  $O(\log n)$ , pero el "log" es log base *phi* (también conocido como proporción áurea, también conocido como  $\sim 1.618$ ).

This is a small but interesting difference—AVL trees offer performance that is not quite as good as perfectly balanced trees since the base is different. But the performance is still very close, since both are  $O(\log n)$  after all. Just know it's not exactly the same.

Ésta es una diferencia pequeña pero interesante: los árboles AVL ofrecen un rendimiento que no es tan bueno como el de los árboles perfectamente equilibrados, ya que la base es diferente. Pero el rendimiento sigue siendo muy similar, ya que, después de todo, ambos son  $O(\log n)$ . Solo sé que no es exactamente lo mismo.

## **Appendix B. NP-hard problems**

---

### **Apéndice B. Problemas NP-difíciles**

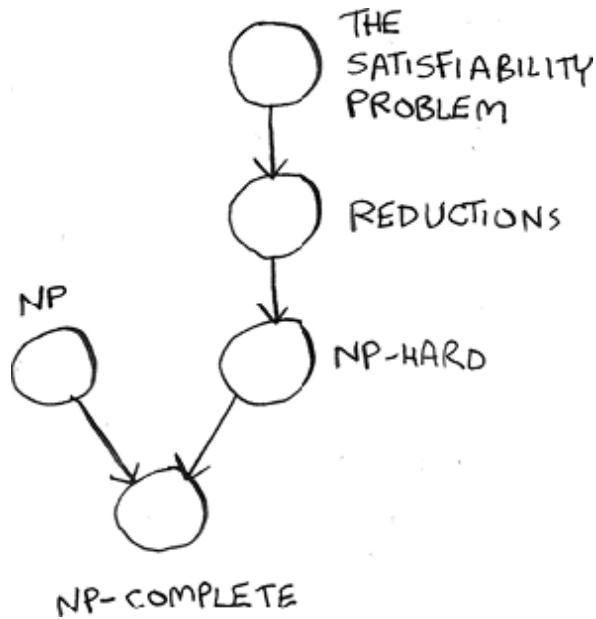
---

Both the set-covering and traveling salesperson problems have something in common: they are hard to solve. You have to check every possible iteration to find the smallest set cover or the shortest route.

Tanto el problema de cubrir los decorados como el del vendedor ambulante tienen algo en común: son difíciles de resolver. Debe verificar todas las iteraciones posibles para encontrar la cobertura del conjunto más pequeño o la ruta más corta.

Both of these problems are NP-hard. The terms *NP*, *NP-hard*, and *NP-complete* can cause a lot of confusion. They certainly confused me. In this appendix, I'll try to explain what all these terms mean, but I need to explain some other concepts first. Here is a roadmap of the things we will learn and how they depend on each other:

Ambos problemas son NP-difíciles. Los términos NP, NP-duro y NP-completo pueden causar mucha confusión. Ciertamente me confundieron. En este apéndice, intentaré explicar qué significan todos estos términos, pero primero necesito explicar algunos otros conceptos. Aquí hay una hoja de ruta de las cosas que aprenderemos y cómo dependen unas de otras:



But, first, I need to explain what a *decision problem* is because all the problems we will look at in the rest of this appendix are decision problems.

Pero primero necesito explicar qué es un problema de decisión porque todos los problemas que veremos en el resto de este apéndice son problemas de decisión.

## ***Decision problems***

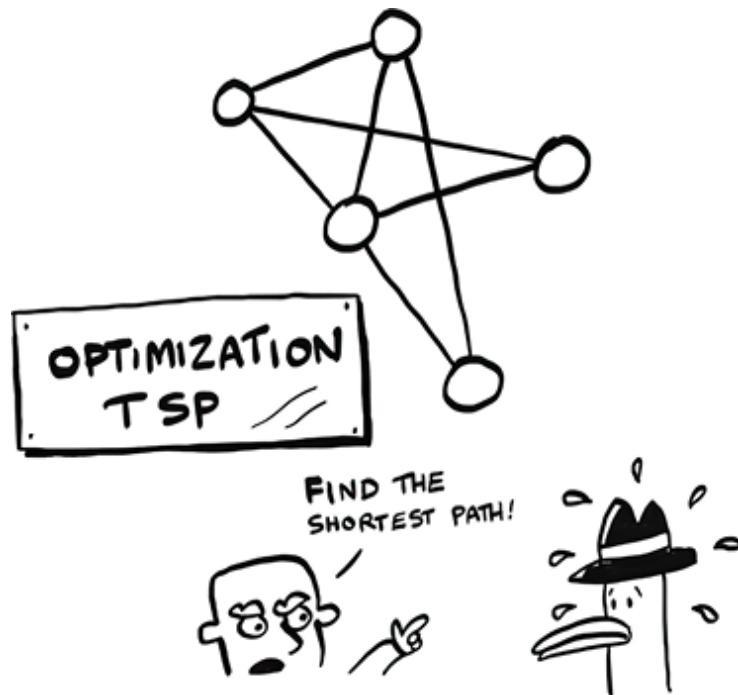
### ***Problemas de decisión***

NP-complete problems are always decision problems. A decision problem has a yes-or-no answer. The traveling salesperson problem is not a decision problem. It's asking you to find the shortest path, which is an optimization problem.

Los problemas NP-completos son siempre problemas de decisión. Un problema de decisión tiene una respuesta de sí o no. El problema del agente ambulante no es un problema de decisión. Le pide que encuentre el camino más corto, lo cual es un problema de optimización.

**NOTE** I know I was talking about *NP-hard* problems in the introduction, and now I'm talking about *NP-complete* problems. I'll explain what the difference is soon.

Tenga en cuenta que sé que estaba hablando de problemas NP-difíciles en la introducción, y ahora estoy hablando de problemas NP-completos. Pronto explicaré cuál es la diferencia.

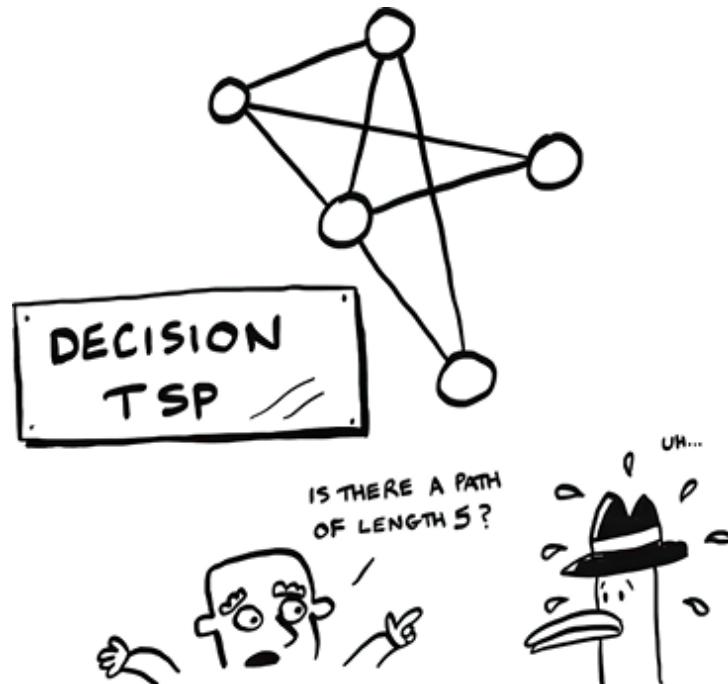


**Find the shortest path!**

**¡Encuentra el camino más corto!**

Here's a decision version of the traveling salesperson problem.

He aquí una versión de decisión del problema del vendedor ambulante.



**Is there a path of length 5?**

**¿Existe un camino de longitud 5?**

Notice how this question has a yes-or-no answer: is there a path of length 5? I wanted to talk about decision problems up front because all NP-complete problems are decision problems. So *all the problems I discuss in the rest of this appendix will be decision problems*. So when you see “traveling salesperson” mentioned in the rest of this appendix, I mean the *decision* version of the traveling salesperson problem.

Observe cómo esta pregunta tiene una respuesta de sí o no: ¿existe un camino de longitud 5? Quería hablar desde el

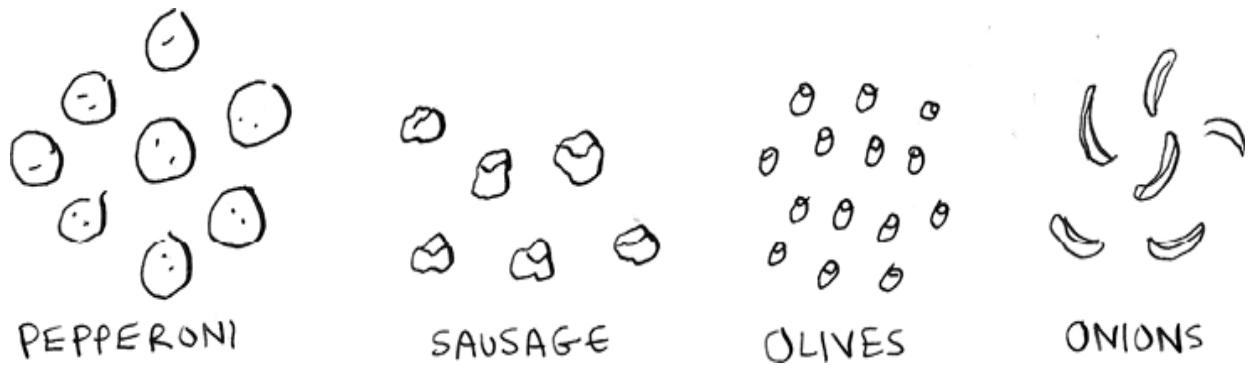
principio sobre los problemas de decisión porque todos los problemas NP-completos son problemas de decisión. Por tanto, todos los problemas que analizo en el resto de este apéndice serán problemas de decisión. Así que cuando veamos la mención del “vendedor ambulante” en el resto de este apéndice, me refiero a la versión de decisión del problema del viajante.

Now let's start learning what NP-complete actually means! The first step is to learn about the satisfiability (SAT) problem.

¡Ahora comencemos a aprender qué significa realmente NP-completo! El primer paso es aprender sobre el problema de satisfacibilidad (SAT).

## ***The satisfiability problem***

## ***El problema de la satisfacibilidad***



Jerry, George, Elaine, and Kramer are all ordering pizza.

Jerry, George, Elaine y Kramer están pidiendo pizza.

"Ooh, let's get pepperoni!" says Elaine.

"¡Oh, vamos por pepperoni!" dice Elaine.

"Pepperoni is good. Sausage is good. We could get pepperoni or sausage," says Jerry.

"El peperoni es bueno. La salchicha es buena. Podríamos conseguir pepperoni o salchichas", dice Jerry.

"Get me an olive pizza to maintain my complexion," says Kramer. "Lots of olives. Or, onions."

"Consígueme una pizza de aceitunas para mantener mi cutis", dice Kramer. "Muchas aceitunas. O cebollas".

"I can do any pizza but *no* onions," says George. "I can't take any more onions, Jerry!"

"Puedo hacer cualquier pizza, pero sin cebolla", dice George.  
"¡No puedo soportar más cebollas, Jerry!"

"Oh boy. OK, let me figure this out. So what toppings do I need again?" says Jerry.

"Oh chico. Bien, déjame resolver esto. Entonces, ¿qué aderezos necesito nuevamente? dice jerry.

Can you help him out? Here are everyone's requirements:

¿Puedes ayudarlo? Aquí están los requisitos de todos:

- Pepperoni (Elaine)
- Pepperoni (Elaine)

- Pepperoni or sausage (Jerry)  
Pepperoni o salchicha (Jerry)
- Olives or onions (Kramer)  
Aceitunas o cebollas (Kramer)
- No onions (George)  
Sin cebollas (George)

See if you can figure out what toppings the pizza should have before moving on.

Intenta descubrir qué ingredientes debe tener la pizza antes de continuar.

Did you get it? A pepperoni and olive pizza satisfies all the requirements. This is an example of a SAT problem. In pseudocode, I could write it like this. First, I have four boolean variables:

¿Lo obtuviste? Una pizza de pepperoni y aceitunas cumple todos los requisitos. Este es un ejemplo de un problema SAT. En pseudocódigo, podría escribirlo así. Primero, tengo cuatro variables booleanas:

```
pepperoni = ?
sausage = ?
olives = ?
onions = ?
```

Then I write out a boolean formula:

Luego escribo una fórmula booleana:

```
(pepperoni) and (pepperoni or sausage) and (olives or onions) and (not onions)
```

This formula contains the requirements for each person in the form of boolean logic. The SAT problem asks the question: Can you set these variables to some values so that the statement evaluates to `true`?

Esta fórmula contiene los requisitos para cada persona en forma de lógica booleana. El problema del SAT plantea la pregunta: ¿Puedes establecer estas variables en algunos valores para que la declaración se evalúe como `true`?

The SAT problem is famous because it is the first NP-complete problem, described in 1971 (although I don't think the authors used Seinfeld as an example). Before this, the concept of an NP-complete problem did not exist. Here is how the SAT problem works. You start with a boolean formula:

El problema SAT es famoso porque es el primer problema NP-completo, descrito en 1971 (aunque no creo que los autores hayan usado Seinfeld como ejemplo). Antes de esto, no existía el concepto de problema NP-completo. Así es como funciona el problema del SAT. Comienzas con una fórmula booleana:

```
if (pepperoni) and (olives or onions):
    print("pizza")
```

Then you ask, is there some way we can assign our variables so that code prints `pizza`?

Entonces preguntas, ¿hay alguna manera de que podamos asignar nuestras variables para que el código imprima

pizza?

This example is pretty easy, so we can solve it ourselves. If pepperoni **and** onions **are** true, this code will print pizza. So the answer would be yes.

Este ejemplo es bastante sencillo, por lo que podemos resolverlo nosotros mismos. Si pepperoni **y** onions **son** true, este código imprimirá pizza. Entonces la respuesta sería sí.

Here's one where the answer would be *no*:

Aquí hay uno donde la respuesta sería no:

```
if (olives or onions) and (not olives) and (not onions):  
    print("pizza")
```

There is nothing you can set the variables to so this code will print pizza!

No hay nada en lo que puedas configurar las variables, por lo que este código imprimirá pizza!



The SAT problem always looks for a yes-or-no answer, so it is a *decision* problem.

El problema del SAT siempre busca una respuesta de sí o no, por lo que es un problema de decisión.

SAT is actually a pretty hard problem. Here is a tougher example just to give you an idea:

El SAT es en realidad un problema bastante difícil. Aquí hay un ejemplo más complicado sólo para darle una idea:

```
if (pepperoni or not olives) and (onions or not pepperoni) and (not olives or not pepperoni):  
    print("pizza")
```

You don't need to solve this one. I'm just showing it as an example so you can appreciate how hard this problem can get. You can have any number of variables and any number of clauses, and the problems get pretty hard pretty quickly.

No es necesario que resuelvas este. Solo lo muestro como un ejemplo para que puedas apreciar lo difícil que puede llegar a ser este problema. Puede tener cualquier cantidad de variables y cláusulas, y los problemas se vuelven bastante difíciles con bastante rapidez.

PEPPERONI	OLIVES	ONIONS	ANSWER
F	F	T	NO
F	T	F	NO
F	T	T	NO
T	F	F	NO
T	F	T	YES
T	T	F	YES
T	T	T	YES
F	F	F	NO

PEPPERONI AND  
(OLIVES OR ONIONS)

With  $n$  toppings, there are  $2^n$  possible pizzas. If you list them all out and check each one, you get something called a truth table. Here's the truth table for pepperoni and (olives or onions).

Con  $n$  ingredientes, hay  $2^n$  pizzas posibles. Si los enumera todos y verifica cada uno, obtendrá algo llamado tabla de verdad. Aquí está la tabla de verdad para pepperoni and (olives or onions).

Sometimes you need to list every option, just like the set-covering problem and the traveling salesperson problem! In fact, the SAT problem is just as hard as these two problems. It has a big O run time of  $O(2^n)$ .

A veces es necesario enumerar todas las opciones, como el problema de cubrir el set y el problema del vendedor ambulante. De hecho, el problema del SAT es tan difícil como estos dos problemas. Tiene un gran tiempo de ejecución O de  $O(2^n)$ .

### ***Hard to solve, quick to verify***

### ***Difícil de resolver, rápido de verificar.***

We often see problems where finding a solution is much harder than verifying a solution. Suppose I ask you to come up with a sentence that is a palindrome (it reads the same backward and forward) that includes the words *cat* and *car*. How long do you think it would take you to come up with that sentence?

A menudo vemos problemas en los que encontrar una solución es mucho más difícil que verificarla. Supongamos que te pido que propongas una oración que sea un

palíndromo (se lee igual hacia adelante y hacia atrás) que incluya las palabras gato y auto. ¿Cuánto tiempo crees que te llevaría llegar a esa frase?

Now suppose I tell you that I know a sentence like that. Here it is: *Was it a car or a cat I saw?*

Ahora supongamos que les digo que conozco una frase como esa. Aquí está: ¿Fue un coche o un gato lo que vi?

It would take you much less time to verify that claim than to come up with your own sentence. Verifying was quicker than solving!

Le llevaría mucho menos tiempo verificar esa afirmación que elaborar su propia frase. ¡Verificar fue más rápido que resolver!

A SAT problem is as hard to solve as the set-covering problem or the traveling salesperson problem, but unlike those problems, verifying a solution is easy. For example, for this question that we gave earlier: (pepperoni or not olives) and (onions or not pepperoni) and (not olives or not pepperoni), here's a solution:

Un problema del SAT es tan difícil de resolver como el problema de la cobertura del set o el problema del vendedor ambulante, pero a diferencia de esos problemas, verificar una solución es fácil. Por ejemplo, para esta pregunta que dimos antes: (pepperoni o no aceitunas) y (cebollas o no pepperoni) y (ni aceitunas o no pepperoni), aquí tienes una solución:

```
pepperoni = False  
olives = False  
onions = False
```

You can quickly check for yourself that these values will make that boolean formula `true`. Checking those values was faster than solving it yourself!

Puedes comprobar rápidamente por ti mismo que estos valores formarán esa fórmula booleana `true`. ¡Verificar esos valores fue más rápido que resolverlo usted mismo!



The SAT problem is *quick to verify*, so it is in NP.

El problema del SAT es rápido de verificar, por eso está en NP.

NP is the class of problems that can be *verified* in polynomial time. NP problems may or may not be easy to solve, but they are easy to verify. This is different from P.

NP es la clase de problemas que se pueden verificar en tiempo polinomial. Los problemas NP pueden ser fáciles o no

de resolver, pero son fáciles de verificar. Esto es diferente de P.



A problem is in P if it can be *verified and solved* in polynomial time.

Un problema está en P si puede verificarse y resolverse en tiempo polinomial.

Polynomial time means its big O is not bigger than a polynomial. I won't define what a polynomial is in this book, but here's two polynomials.

El tiempo polinómico significa que su gran O no es mayor que un polinomio. No definiré qué es un polinomio en este libro, pero aquí hay dos polinomios.

$$n^3 \quad n^2 + n$$

And here are a couple of examples that are not polynomials.

Y aquí hay un par de ejemplos que no son polinomios.

$$n! \quad 2^n$$

P is a subset of NP. So NP contains all the problems in P, plus others.

P es un subconjunto de NP. Entonces NP contiene todos los problemas de P, más otros.



## P vs. NP

### P frente a NP

You may have heard of the famous *P versus NP* problem. We just saw that the problems in P are both quick to verify and quick to solve. The problems in NP are quick to verify but may or may not be quick to solve. The P versus NP problem asks whether every problem that is quick to verify *is also quick to solve*. If that is the case, P wouldn't be a subset of NP; P would equal NP.

Es posible que hayas oído hablar del famoso problema P versus NP. Acabamos de ver que los problemas en P son rápidos de verificar y de resolver. Los problemas en NP se verifican rápidamente, pero pueden resolverse rápidamente o no. El problema P versus NP pregunta si todo problema que es rápido de verificar también es rápido de resolver. Si ese es el caso, P no sería un subconjunto de NP; P sería igual a NP.

NP-hard is the next term we will define, but first we need to (briefly) discuss what a reduction is.

NP-duro es el siguiente término que definiremos, pero primero debemos discutir (brevemente) qué es una reducción.

## **Reductions**

### **Reducciones**

What do you do when you have a hard problem? Change the problem to one you can solve! In real life, when we're faced with a hard problem, it is extremely common to change the problem.

¿Qué haces cuando tienes un problema difícil? ¡Cambia el problema por uno que puedas resolver! En la vida real, cuando nos enfrentamos a un problema difícil, es muy común cambiarlo.

Here's one you can try right now. How do you multiply two binary numbers? Try multiplying these two binary numbers:

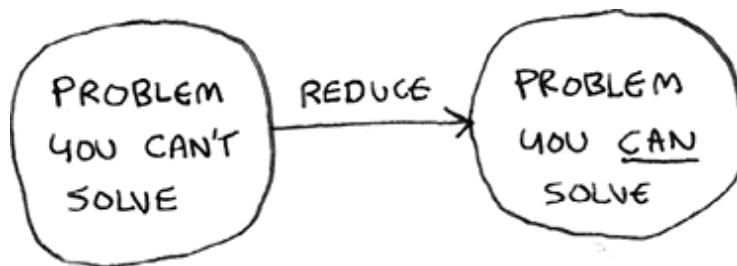
Aquí tienes uno que puedes probar ahora mismo. ¿Cómo se multiplican dos números binarios? Intenta multiplicar estos dos números binarios:

If you're like me, you didn't try to figure out how to do the multiplication in binary. You just figured out that  $101$  is 5 in decimal and  $110$  is 6, and then you multiplied 5 and 6 instead.

Si eres como yo, no intentaste descubrir cómo hacer la multiplicación en binario. Acabas de descubrir que  $101$  es 5 en decimal y  $110$  es 6, y luego multiplicaste 5 y 6.

This is called a reduction. You are reducing a problem that you don't know how to solve to a problem you do know how to solve. This is done all the time in computer science.

Esto se llama reducción. Estás reduciendo un problema que no sabes cómo resolver a un problema que sí sabes cómo resolver. Esto se hace todo el tiempo en informática.



## **NP-hard**

## **NP-duro**

We have already seen three examples of NP-hard problems:

Ya hemos visto tres ejemplos de problemas NP-difíciles:

- The set-covering problem  
El problema de la cobertura de conjuntos
- The traveling salesperson problem  
El problema del vendedor ambulante
- The SAT problem  
El problema del SAT

(Remember, I mean the *decision* versions of these problems —all the problems we look at in the rest of this appendix are decision problems.)

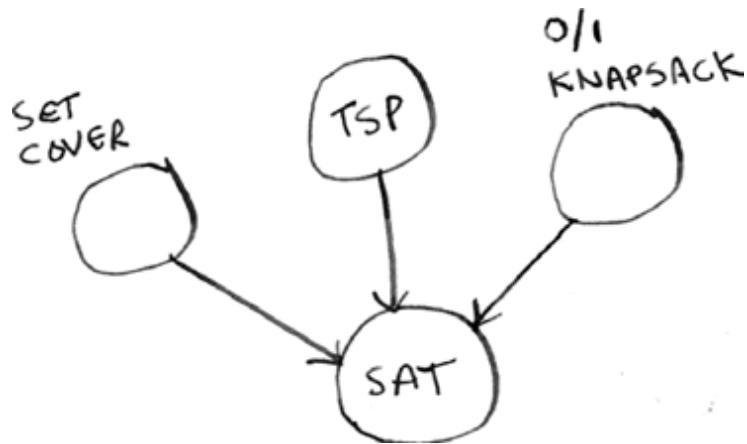
(Recuerde, me refiero a las versiones de decisión de estos problemas; todos los problemas que analizamos en el resto de este apéndice son problemas de decisión).

The three previously noted problems are NP-hard. We say a problem is NP-hard *if any problem in NP can be reduced to that problem*. This is the definition of NP-hard.

Los tres problemas mencionados anteriormente son NP-difíciles. Decimos que un problema es NP-difícil si algún problema en NP puede reducirse a ese problema. Esta es la definición de NP-duro.

You can also reduce all NP problems to any NP-hard problem. For example, you can reduce all NP problems to SAT.

También puede reducir todos los problemas NP a cualquier problema NP-difícil. Por ejemplo, puedes reducir todos los problemas NP a SAT.



One extra requirement is that you need to be able to reduce all these problems *in polynomial time*. That “in polynomial time” is important because you don’t want the reducing part to be the bottleneck. Any NP problem can be reduced to SAT in polynomial time, so it is NP-hard.

Un requisito adicional es que debe poder reducir todos estos problemas en tiempo polinomial. Eso “en tiempo polinómico” es importante porque no desea que la parte reductora sea el cuello de botella. Cualquier problema NP se puede reducir a SAT en tiempo polinomial, por lo que es NP-difícil.

Since any problem in NP can be reduced to any NP-hard problem, a polynomial time solution for any one NP-hard problem gives us a polynomial time solution for every problem in NP!

Dado que cualquier problema en NP se puede reducir a cualquier problema NP-difícil, una solución en tiempo polinomial para cualquier problema NP-difícil nos da una solución en tiempo polinomial para cada problema en NP.

## **NP-complete**

### **NP-completo**

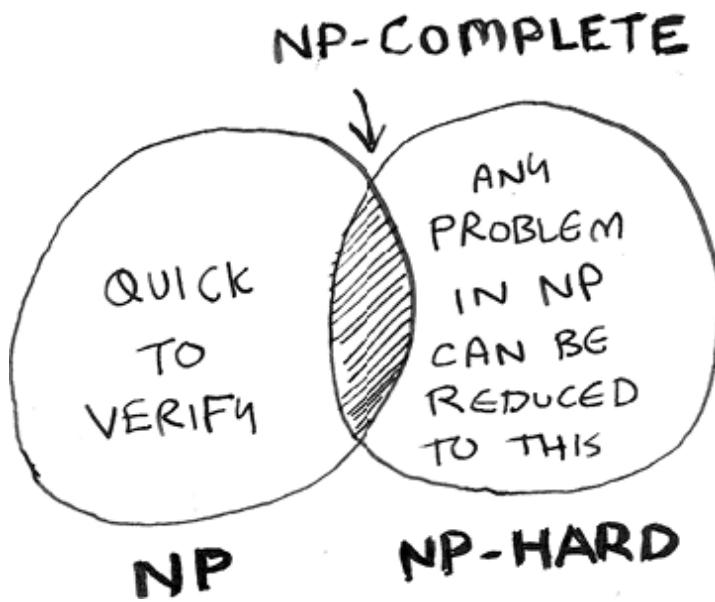
We've seen two definitions:

Hemos visto dos definiciones:

- Problems in NP are quick to verify and may or may not be quick to solve.  
Los problemas en NP se verifican rápidamente y pueden resolverse rápidamente o no.
- Problems that are NP-hard are at least as hard as the hardest problems in NP, and any problem in NP can be reduced to a problem in NP-hard.  
Los problemas que son NP-difíciles son al menos tan difíciles como los problemas más difíciles en NP, y cualquier problema en NP puede reducirse a un problema en NP-difícil.

Now here's my final definition: *a problem is NP-complete if it is both NP and NP-hard.*

Ahora aquí está mi definición final: un problema es NP-completo si es tanto NP como NP-difícil.



NP-complete problems are

Los problemas NP-completos son

- Hard to solve (at least right now; if someone proves that  $P = NP$ , they would not be)  
Difícil de resolver (al menos ahora; si alguien demuestra que  $P = NP$ , no lo sería)
- Easy to verify  
Fácil de verificar

And any problem in NP can be reduced to a problem that is NP-complete.

Y cualquier problema en NP puede reducirse a un problema NP completo.

Here are the terms we defined in this appendix:

Estos son los términos que definimos en este apéndice:

- Decision problems  
Problemas de decisión
- The SAT problem  
El problema del SAT
- P versus NP  
P versus NP
- Reductions  
Reducciones
- NP-hard  
NP-duro
- NP-complete  
NP-completo

When you see a discussion about NP-complete problems, I hope you'll feel more confident about what these terms mean!

Cuando vea una discusión sobre problemas NP-completos, espero que se sienta más seguro acerca de lo que significan estos términos.

## **Recap**

### **Resumen**

- A problem is in P if it is both quick to solve and quick to verify.

Un problema está en P si es rápido de resolver y de verificar.

- A problem is in NP if it is quick to verify. It may or may not be quick to solve.

Un problema está en NP si se verifica rápidamente.  
Puede que su solución sea rápida o no.

- If we find a fast (polynomial time) algorithm for every problem in NP, then  $P = NP$ .

Si encontramos un algoritmo rápido (tiempo polinómico) para cada problema en NP, entonces  $P = NP$ .

- A problem is NP-hard if any problem in NP can be reduced to that problem.

Un problema es NP-difícil si cualquier problema en NP puede reducirse a ese problema.

- If a problem is in both NP and NP-hard, it is NP-complete.

Si un problema está tanto en NP como en NP-duro, es NP-completo.

## **Appendix C. Answers to exercises**

---

## **Apéndice C. Respuestas a los ejercicios**

---

### **CHAPTER 1**

#### **CAPÍTULO 1**

**1.1** Suppose you have a sorted list of 128 names, and you're searching through it using binary search. What's the maximum number of steps it would take?

1.1 Suponga que tiene una lista ordenada de 128 nombres y está buscando en ella mediante búsqueda binaria. ¿Cuál es el número máximo de pasos que daría?

*Answer:* 7.

Respuesta: 7.

**1.2** Suppose you double the size of the list. What's the maximum number of steps now?

1.2 Suponga que duplica el tamaño de la lista. ¿Cuál es el número máximo de pasos ahora?

*Answer:* 8.

Respuesta: 8.

**1.3** You have a name, and you want to find the person's phone number in the phone book.

1.3 Tiene un nombre y desea encontrar el número de teléfono de la persona en la guía telefónica.

*Answer:*  $O(\log n)$ .

Respuesta:  $O(\log n)$ .

**1.4** You have a phone number, and you want to find the person's name in the phone book. (Hint: You'll have to search through the whole book!)

1.4 Tiene un número de teléfono y desea encontrar el nombre de la persona en la guía telefónica. (Pista: itendrás que buscar en todo el libro!)

*Answer:*  $O(n)$ .

Respuesta:  $O(n)$ .

**1.5** You want to read the numbers of every person in the phone book.

1.5 Quiere leer los números de cada persona en la guía telefónica.

*Answer:*  $O(n)$ .

Respuesta:  $O(n)$ .

**1.6** You want to read the numbers of just the As.

## 1.6 Quieres leer los números solo de las As.

*Answer:*  $O(n)$ . You may think, "I'm only doing this for 1 out of 26 characters, so the run time should be  $O(n/26)$ ." A simple rule to remember is to ignore numbers that are added, subtracted, multiplied, or divided. None of these are correct big O run times:  $O(n + 26)$ ,  $O(n - 26)$ ,  $O(n * 26)$ ,  $O(n/26)$ . They're all the same as  $O(n)$ ! Why? If you're curious, flip to "big O notation revisited" in chapter 4 and read up on constants in big O notation (a constant is just a number; 26 was the constant in this question).

**Respuesta:**  $O(n)$ . Quizás pienses: "Solo estoy haciendo esto para 1 de 26 caracteres, por lo que el tiempo de ejecución debería ser  $O(n/26)$ ". Una regla simple para recordar es ignorar los números que se suman, restan, multiplican o dividen. Ninguno de estos son tiempos de ejecución correctos de O grande:  $O(n + 26)$ ,  $O(n - 26)$ ,  $O(n * 26)$ ,  $O(n/26)$ . ¡Son todos iguales que  $O(n)$ ! ¿Por qué? Si tiene curiosidad, pase a "Revisión de la notación O grande" en el capítulo 4 y lea sobre las constantes en notación O grande (una constante es solo un número; 26 era la constante en esta pregunta).

## CHAPTER 2

### CAPITULO 2

**2.1** Suppose you're building an app to keep track of your finances.

2.1 Suponga que está creando una aplicación para realizar un seguimiento de sus finanzas.

1. GROCERIES
2. MOVIE
3. SFBC  
MEMBERSHIP

Every day, you write down everything you spent money on. At the end of the month, you review your expenses and sum up how much you spent. So, you have lots of inserts and a few reads. Should you use an array or a list?

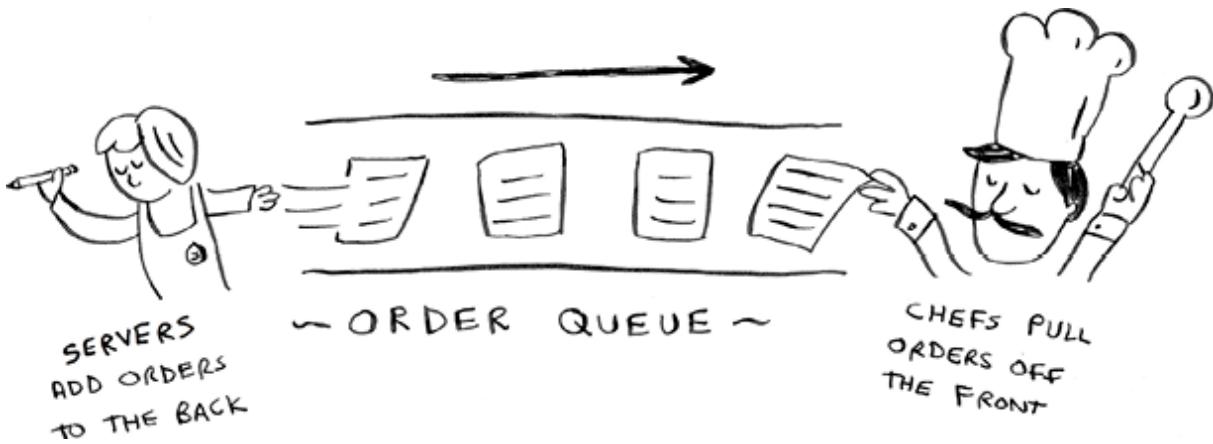
Todos los días, anotas todo en lo que gastaste dinero. Al final del mes, revisas tus gastos y sumas cuánto gastaste. Entonces, tienes muchos encartes y algunas lecturas. ¿Deberías usar una matriz o una lista?

*Answer:* In this case, you're adding expenses to the list every day and reading all the expenses once a month. Arrays have fast reads and slow inserts. Linked lists have slow reads and fast inserts. Because you'll be inserting more often than reading, it makes sense to use a linked list. Also, linked lists have slow reads only if you're accessing random elements in the list. Because you're reading *every* element in the list, linked lists will do well on *reads*, too. So a linked list is a good solution to this problem.

Respuesta: En este caso, agrega gastos a la lista todos los días y lee todos los gastos una vez al mes. Las matrices tienen lecturas rápidas e inserciones lentas. Las listas enlazadas tienen lecturas lentas e inserciones rápidas. Debido a que insertará más veces que leerá, tiene sentido utilizar una lista vinculada. Además, las listas vinculadas tienen lecturas lentas solo si accede a elementos aleatorios de la lista. Debido a que estás leyendo cada elemento de la lista, las listas vinculadas también funcionarán bien en las lecturas. Entonces una lista enlazada es una buena solución a este problema.

**2.2** Suppose you're building an app for restaurants to take customer orders. Your app needs to store a list of orders. Servers keep adding orders to this list, and chefs take orders off the list and make them. It's an order queue: servers add orders to the back of the queue, and the chef takes the first order off the queue and cooks it.

2.2 Suponga que está creando una aplicación para que los restaurantes tomen pedidos de los clientes. Tu aplicación necesita almacenar una lista de pedidos. Los camareros siguen agregando pedidos a esta lista y los chefs quitan pedidos de la lista y los preparan. Es una cola de pedidos: los servidores agregan pedidos al final de la cola y el chef toma el primer pedido de la cola y lo cocina.



Would you use an array or a linked list to implement this queue? (Hint: Linked lists are good for inserts/deletes, and arrays are good for random access. Which one are you going to be doing here?)

¿Usarías una matriz o una lista vinculada para implementar esta cola? (Sugerencia: las listas vinculadas son buenas para insertar/eliminar, y las matrices son buenas para el acceso aleatorio. ¿Cuál vas a hacer aquí?)

*Answer:* A linked list. Lots of inserts are happening (servers adding orders), which linked lists excel at. You don't need search or random access (what arrays excel at) because the chefs always take the first order off the queue.

*Respuesta:* Una lista enlazada. Se están produciendo muchas inserciones (servidores agregando pedidos), en las que se destacan las listas vinculadas. No necesita búsqueda ni acceso aleatorio (en qué destacan los arreglos) porque los chefs siempre toman el primer pedido de la cola.

**2.3** Let's run a thought experiment. Suppose Facebook keeps a list of usernames. When someone tries to log in to Facebook, a search is done for their username. If their name is in the list of usernames, they can log in. People log in to Facebook pretty often, so there are a lot of searches through this list of usernames. Suppose Facebook uses binary search to search the list. Binary search needs random access—you need to be able to get to the middle of the list of usernames instantly. Knowing this, would you implement the list as an array or a linked list?

2.3 Hagamos un experimento mental. Supongamos que Facebook mantiene una lista de nombres de usuario. Cuando alguien intenta iniciar sesión en Facebook, se realiza una búsqueda de su nombre de usuario. Si su nombre está en la lista de nombres de usuario, pueden iniciar sesión. La gente inicia sesión en Facebook con bastante frecuencia, por lo que hay muchas búsquedas en esta lista de nombres de usuario. Supongamos que Facebook usa la búsqueda binaria para buscar en la lista. La búsqueda binaria necesita acceso aleatorio: debe poder llegar al centro de la lista de nombres de usuario al instante. Sabiendo esto, ¿implementarías la lista como una matriz o una lista enlazada?

*Answer:* A sorted array. Arrays give you random access—you can get an element from the middle of the array instantly. You can't do that with linked lists. To get to the middle element in a linked list, you'd

have to start at the first element and follow all the links down to the middle element.

Respuesta: una matriz ordenada. Las matrices le brindan acceso aleatorio: puede obtener un elemento del centro de la matriz al instante. No puedes hacer eso con listas vinculadas. Para llegar al elemento del medio en una lista vinculada, tendría que comenzar en el primer elemento y seguir todos los enlaces hasta el elemento del medio.

**2.4** People sign up for Facebook pretty often, too. Suppose you decided to use an array to store the list of users. What are the downsides of an array for inserts? In particular, suppose you're using binary search to search for logins. What happens when you add new users to an array?

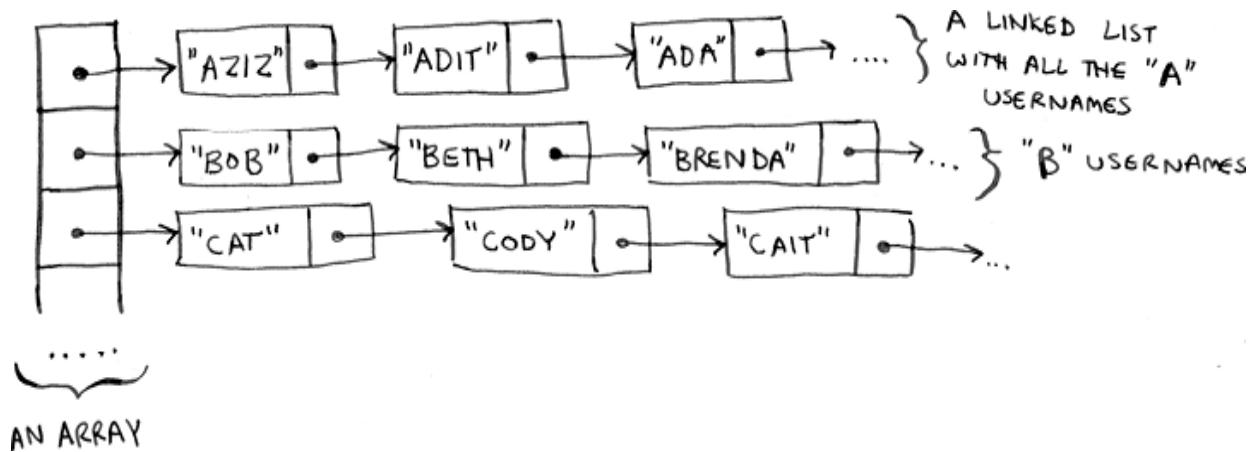
2.4 La gente también se registra en Facebook con bastante frecuencia. Suponga que decide utilizar una matriz para almacenar la lista de usuarios. ¿Cuáles son las desventajas de una matriz para inserciones? En particular, supongamos que está utilizando la búsqueda binaria para buscar inicios de sesión. ¿Qué sucede cuando agrega nuevos usuarios a una matriz?

*Answer:* Inserting into arrays is slow. Also, if you're using binary search to search for usernames, the array needs to be sorted. Suppose someone named Adit B signs up for Facebook. Their name will be inserted at the end of the array. So you need to sort the array every time a name is inserted!

Respuesta: La inserción en matrices es lenta. Además, si utiliza la búsqueda binaria para buscar nombres de usuario, es necesario ordenar la matriz. Supongamos que alguien llamado Adit B se registra en Facebook. Su nombre se insertará al final de la matriz. ¡Por lo tanto, debes ordenar la matriz cada vez que se inserta un nombre!

**2.5** In reality, Facebook uses neither an array nor a linked list to store user information. Let's consider a hybrid data structure: an array of linked lists. You have an array with 26 slots. Each slot points to a linked list. For example, the first slot in the array points to a linked list containing all the usernames starting with *A*. The second slot points to a linked list containing all the usernames starting with *B*, and so on.

2.5 En realidad, Facebook no utiliza ni una matriz ni una lista enlazada para almacenar información del usuario. Consideremos una estructura de datos híbrida: una serie de listas enlazadas. Tienes una matriz con 26 ranuras. Cada ranura apunta a una lista vinculada. Por ejemplo, el primer espacio de la matriz apunta a una lista vinculada que contiene todos los nombres de usuario que comienzan con *A*. El segundo espacio apunta a una lista vinculada que contiene todos los nombres de usuario que comienzan con *B*, y así sucesivamente.



Suppose Adit B signs up for Facebook, and you want to add them to the list. You go to slot 1 in the array, go to the linked list for slot 1, and add Adit B at the end. Now, suppose you want to search for Zakhir H. You go to slot 26, which points to a linked list of all the Z names. Then you search through that list to find Zakhir H.

Supongamos que Adit B se registra en Facebook y desea agregarlo a la lista. Vaya a la ranura 1 de la matriz, vaya a la lista vinculada para la ranura 1 y agregue Adit B al final. Ahora, supongamos que desea buscar a Zakhir H. Vaya al espacio 26, que apunta a una lista vinculada de todos los nombres Z. Luego buscas en esa lista para encontrar a Zakhir H.

Compare this hybrid data structure to arrays and linked lists. Is it slower or faster than each for searching and inserting? You don't have to give big O run times, just whether the new data structure would be faster or slower.

Compare esta estructura de datos híbrida con matrices y listas vinculadas. ¿Es más lento o más rápido que cada uno para buscar e insertar? No es necesario indicar tiempos de ejecución O grandes, solo si la nueva estructura de datos sería más rápida o más lenta.

*Answer:* Searching—slower than arrays, faster than linked lists. Inserting—faster than arrays, same amount of time as linked lists. So it's slower for searching than an array, but faster or the same as linked lists for everything. We'll talk about another hybrid data structure called a hash table later in the book. This should give you an idea of how you can build up more complex data structures from simple ones.

Respuesta: Búsqueda: más lenta que las matrices, más rápida que las listas vinculadas. Insertar: más rápido que las matrices y en el mismo tiempo que las listas enlazadas. Por lo tanto, es más lento para buscar que una matriz, pero más rápido o igual que las listas vinculadas para todo. Hablaremos de otra estructura de datos híbrida llamada tabla hash más adelante en el libro. Esto debería darle una idea de cómo puede construir estructuras de datos más complejas a partir de estructuras simples.

So what does Facebook really use? It probably uses a dozen different databases with different data structures behind them: hash tables, B-trees, and

others. Arrays and linked lists are the building blocks for these more complex data structures.

Entonces, ¿qué utiliza realmente Facebook?  
Probablemente utilice una docena de bases de datos diferentes con diferentes estructuras de datos detrás de ellas: tablas hash, árboles B y otros. Las matrices y las listas enlazadas son los componentes básicos de estas estructuras de datos más complejas.

## CHAPTER 3

### CAPÍTULO 3

**3.1** Suppose I show you a call stack like this.

3.1 Supongamos que les muestro una pila de llamadas como esta.



What information can you give me, just based on this call stack?

¿Qué información puede darme, basándose únicamente en esta pila de llamadas?

*Answer:* Here are some things you could tell me:

Respuesta: Aquí hay algunas cosas que podrías decirme:

The `greet` function is called first, with `name = maggie`.

La función `greet` se llama primero, con `name = maggie`.

Then the `greet` function calls the `greet2` function, with `name = maggie`.

Luego la función `greet` llama a la función `greet2`, con `name = maggie`.

At this point, the `greet` function is in an incomplete, suspended state.

En este punto, la función `greet` está en un estado suspendido e incompleto.

The current function call is the `greet2` function.

La llamada de función actual es la función `greet2`.

After this function call completes, the `greet` function will resume.

Una vez que se complete esta llamada de función, la función `greet` se reanudará.

**3.2** Suppose you accidentally write a recursive function that runs forever. As you saw, your

computer allocates memory on the stack for each function call. What happens to the stack when your recursive function runs forever?

3.2 Suponga que accidentalmente escribe una función recursiva que se ejecuta para siempre. Como vio, su computadora asigna memoria en la pila para cada llamada de función. ¿Qué sucede con la pila cuando su función recursiva se ejecuta para siempre?

*Answer:* The stack grows forever. Each program has a limited amount of space on the call stack. When your program runs out of space (which it eventually will), it will exit with a stack-overflow error.

Respuesta: La pila crece para siempre. Cada programa tiene una cantidad limitada de espacio en la pila de llamadas. Cuando su programa se quede sin espacio (lo que eventualmente sucederá), saldrá con un error de desbordamiento de pila.

## CHAPTER 4

### CAPÍTULO 4

**4.1** Write out the code for the earlier sum function.

4.1 Escriba el código para la función de suma anterior.

*Answer:*

Respuesta:

```
def sum(list):
    if list == []:
        return 0
    return list[0] + sum(list[1:])
```

**4.2** Write a recursive function to count the number of items in a list.

4.2 Escriba una función recursiva para contar el número de elementos en una lista.

*Answer:*

Respuesta:

```
def count(list):
    if list == []:
        return 0
    return 1 + count(list[1:])
```

**4.3** Write a recursive function to find the maximum number in a list.

4.3 Escriba una función recursiva para encontrar el número máximo en una lista.

*Answer:*

Respuesta:

```
def max(list):
    if len(list) == 2:
        return list[0] if list[0] > list[1] else list[1]
    sub_max = max(list[1:])
    return list[0] if list[0] > sub_max else sub_max
```

**4.4** Remember binary search from chapter 1? It's a D&C algorithm, too. Can you come up with the base

case and recursive case for binary search?

4.4 ¿Recuerdas la búsqueda binaria del capítulo 1?

También es un algoritmo D&C. ¿Puedes proponer el caso base y el caso recursivo para la búsqueda binaria?

**Answer:** The base case for binary search is an array with one item. If the item you're looking for matches the item in the array, you found it! Otherwise, it isn't in the array.

**Respuesta:** El caso base para la búsqueda binaria es una matriz con un elemento. Si el elemento que busca coincide con el elemento de la matriz, lo encontró! De lo contrario, no estará en la matriz.

In the recursive case for binary search, you split the array in half, throw away one half, and call binary search on the other half.

En el caso recursivo de la búsqueda binaria, se divide la matriz por la mitad, se desecha una mitad y se llama a la búsqueda binaria en la otra mitad.

How long would each of these operations take in big O notation?

¿Cuánto tiempo tomaría cada una de estas operaciones en notación O grande?

**4.5** Printing the value of each element in an array.

4.5 Imprimir el valor de cada elemento de un array.

*Answer:* O(n)

Respuesta: O(n)

**4.6** Doubling the value of each element in an array.

4.6 Duplicar el valor de cada elemento de una matriz.

*Answer:* O(n)

Respuesta: O(n)

**4.7** Doubling the value of just the first element in an array.

4.7 Duplicar el valor de solo el primer elemento de una matriz.

*Answer:* O(1)

Respuesta: O(1)

**4.8** Creating a multiplication table with all the elements in the array. So if your array is [2, 3, 7, 8, 10], you first multiply every element by 2, then multiply every element by 3, then by 7, and so on.

4.8 Creando una tabla de multiplicar con todos los elementos de la matriz. Entonces, si tu matriz es [2, 3, 7, 8, 10], primero multiplicas cada elemento por

2, luego multiplicas cada elemento por 3, luego por 7, y así sucesivamente.

*Answer:*  $O(n^2)$

Respuesta:  $O(n^2)$

## CHAPTER 5

### CAPÍTULO 5

Which of these hash functions are consistent?

¿Cuáles de estas funciones hash son consistentes?

**5.1**  $f(x) = 1$  ← Returns 1 for all input

5.1  $f(x) = 1$  ← Devuelve 1 para todas las entradas

*Answer:* Consistent

Respuesta: consistente

**5.2**  $f(x) = \text{rand}()$  ← Returns a random number every time

5.2  $f(x) = \text{rand}()$  ← Devuelve un número aleatorio cada vez

*Answer:* Not consistent

Respuesta: No consistente

**5.3**  $f(x) = \text{next\_empty\_slot}()$  ← Returns the index of the next empty slot in the hash table

5.3  $f(x) = \text{next\_empty\_slot}()$  ← Devuelve el índice del siguiente espacio vacío en la tabla hash

*Answer:* Not consistent

Respuesta: No consistente

**5.4**  $f(x) = \text{len}(x)$  ← Uses the length of the string as the index

5.4  $f(x) = \text{len}(x)$  ← Utiliza la longitud de la cadena como índice

*Answer:* Consistent

Respuesta: consistente

Suppose you have these four hash functions that work with strings:

Suponga que tiene estas cuatro funciones hash que funcionan con cadenas:

- A. Return “1” for all input.
- A. Devuelve “1” para todas las entradas.
- B. Use the length of the string as the index.
- B. Utilice la longitud de la cadena como índice.

C. Use the first character of the string as the index. So, all strings starting with *a* are hashed together, and so on.

C. Utilice el primer carácter de la cadena como índice. Entonces, todas las cadenas que comienzan con *a* se combinan, y así sucesivamente.

D. Map every letter to a prime number:  $a = 2, b = 3, c = 5, d = 7, e = 11$ , and so on. For a string, the hash function is the sum of all the characters modulo the size of the hash. For example, if your hash size is 10, and the string is "bag," the index is  $3 + 2 + 17 \% 10 = 22 \% 10 = 2$ .

D. Asigne cada letra a un número primo:  $a = 2, b = 3, c = 5, d = 7, e = 11$ , etc. Para una cadena, la función hash es la suma de todos los caracteres módulo del tamaño del hash. Por ejemplo, si el tamaño de su hash es 10 y la cadena es "bolsa", el índice es  $3 + 2 + 17 \% 10 = 22 \% 10 = 2$ .

For each of the following examples, which hash functions would provide a good distribution? Assume a hash table size of 10 slots.

Para cada uno de los siguientes ejemplos, ¿qué funciones hash proporcionarían una buena distribución? Supongamos un tamaño de tabla hash de 10 espacios.

**5.5** A phonebook where the keys are names and values are phone numbers. The names are as

follows: Esther, Ben, Bob, and Dan.

5.5 Una agenda donde las claves son nombres y los valores son números de teléfono. Los nombres son los siguientes: Esther, Ben, Bob y Dan.

*Answer:* Hash functions C and D would give a good distribution.

Respuesta: Las funciones hash C y D darían una buena distribución.

**5.6** A mapping from battery size to power. The sizes are A, AA, AAA, and AAAA.

5.6 A mapeo del tamaño de la batería a la potencia. Los tamaños son A, AA, AAA y AAAA.

*Answer:* Hash functions B and D would give a good distribution.

Respuesta: Las funciones hash B y D darían una buena distribución.

**5.7** A mapping from book titles to authors. The titles are *Maus*, *Fun Home*, and *Watchmen*.

5.7 Un mapeo desde los títulos de los libros hasta los autores. Los títulos son *Maus*, *Fun Home* y *Watchmen*.

*Answer:* Hash functions B, C, and D would give a good distribution.

Respuesta: Las funciones hash B, C y D darían una buena distribución.

## CHAPTER 6

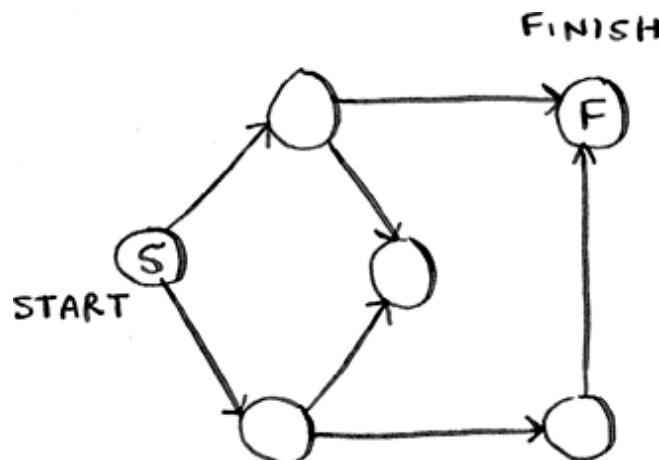
### CAPÍTULO 6

Run the breadth-first search algorithm on each of these graphs to find the solution.

Ejecute el algoritmo de búsqueda en amplitud en cada uno de estos gráficos para encontrar la solución.

**6.1** Find the length of the shortest path from start to finish.

6.1 Encuentre la longitud del camino más corto de principio a fin.

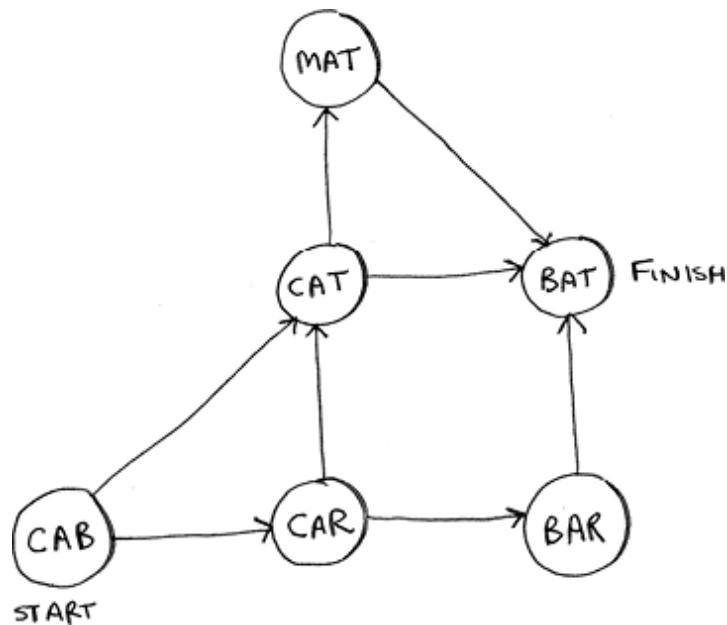


Answer: The shortest path has a length of 2.

Respuesta: El camino más corto tiene una longitud de 2.

**6.2** Find the length of the shortest path from "cab" to "bat."

6.2 Encuentre la longitud del camino más corto desde "taxi" hasta "bat".

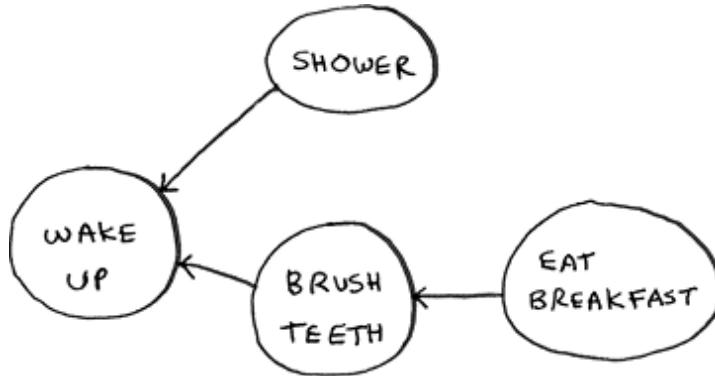


*Answer:* The shortest path has a length of 2.

Respuesta: El camino más corto tiene una longitud de 2.

Here's a small graph of my morning routine.

Aquí tenéis un pequeño gráfico de mi rutina matutina.



**6.3** For these three lists, mark whether each one is valid or invalid.

6.3 Para estas tres listas, marque si cada una es válida o no válida.

A.

1. WAKE UP
2. SHOWER
3. EAT BREAKFAST
4. BRUSH TEETH

B.

1. WAKE UP
2. BRUSH TEETH
3. EAT BREAKFAST
4. SHOWER

C.

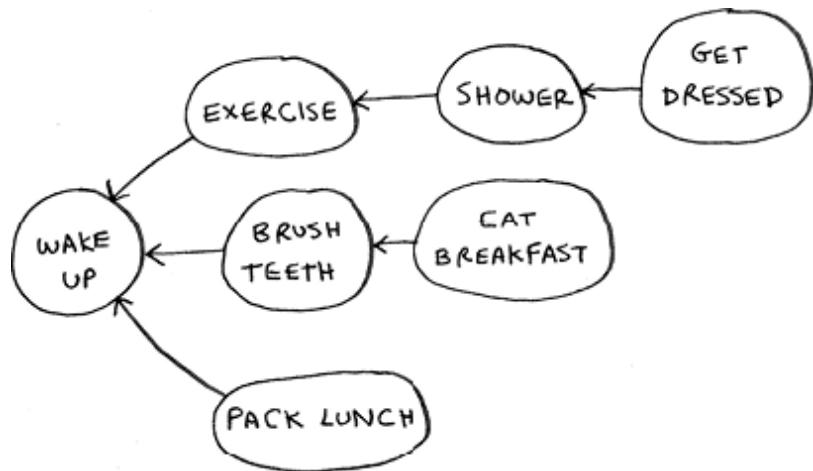
1. SHOWER
2. WAKE UP
3. BRUSH TEETH
4. EAT BREAKFAST

Answers: A—Invalid; B—Valid; C—Invalid.

Respuestas: A—No válido; B—Válido; C—No válido.

**6.4** Here's a larger graph. Make a valid list for this graph.

6.4 Aquí hay un gráfico más grande. Haz una lista válida para este gráfico.



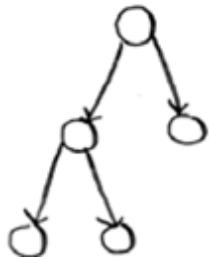
*Answer:* 1—Wake up; 2—Exercise; 3—Shower; 4—Brush teeth; 5—Get dressed; 6—Pack lunch; 7—Eat breakfast.

*Respuesta:* 1—Despierta; 2—Ejercicio; 3—Ducha; 4—Cepillarse los dientes; 5—Vístete; 6—Llevar el almuerzo; 7—Desayuna.

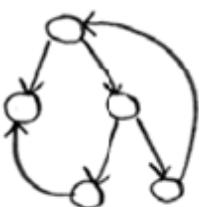
**6.5** Which of the following graphs are also trees?

6.5 ¿Cuáles de los siguientes gráficos también son árboles?

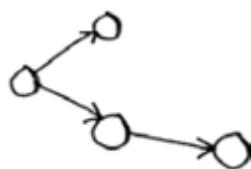
A.



B.



C.



*Answers:* A—Tree; B—Not a tree; C—Tree. The last example is just a sideways tree. Trees are a subset of graphs. So a tree is always a graph, but a graph may or may not be a tree.

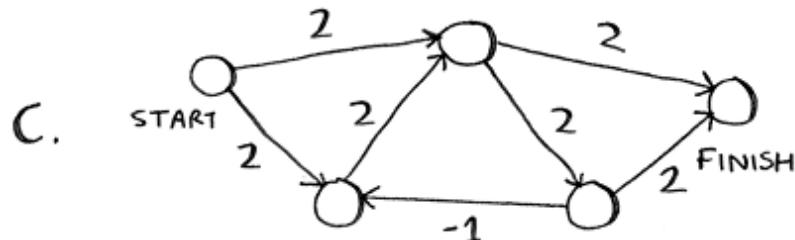
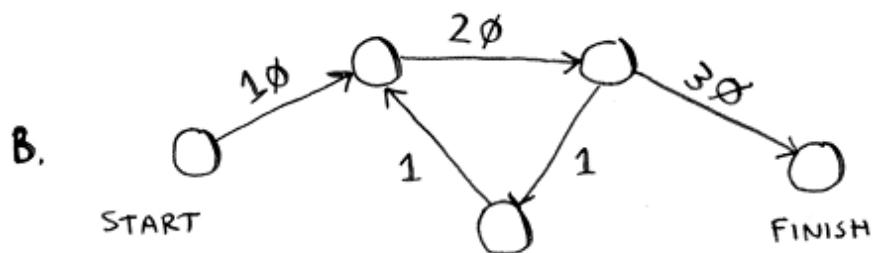
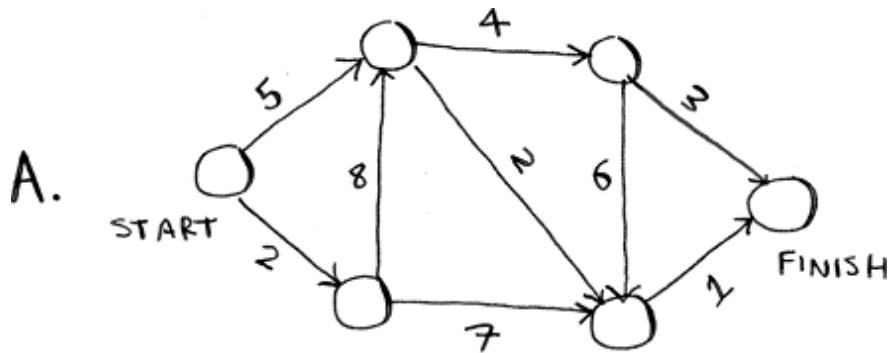
*Respuestas:* A—Árbol; B—No es un árbol; C—árbol. El último ejemplo es sólo un árbol de lado. Los árboles son un subconjunto de gráficos. Entonces, un árbol es siempre un gráfico, pero un gráfico puede ser un árbol o no.

## CHAPTER 9

### CAPÍTULO 9

**9.1** In each of these graphs, what is the weight of the shortest path from Start to Finish?

9.1 En cada uno de estos gráficos, ¿cuál es el peso del camino más corto desde el principio hasta el final?



Answers: A: A—8; B—60; C—4.

Respuestas: A: A—8; B—60; C—4.

## CHAPTER 10

### CAPÍTULO 10

**10.1** You work for a furniture company, and you have to ship furniture all over the country. You need to pack your truck with boxes. All the boxes are of different sizes, and you're trying to maximize the

space you use in each truck. How would you pick boxes to maximize space? Come up with a greedy strategy. Will that give you the optimal solution?

10.1 Trabajas para una empresa de muebles y tienes que enviar muebles a todo el país. Necesitas empacar tu camión con cajas. Todas las cajas son de diferentes tamaños e intentas maximizar el espacio que utilizas en cada camión. ¿Cómo elegirías cajas para maximizar el espacio? Piensa en una estrategia codiciosa. ¿Eso le dará la solución óptima?

*Answer:* A greedy strategy would be to pick the largest box that will fit in the remaining space and repeat until you can't pack any more boxes. No, this won't give you the optimal solution.

Respuesta: Una estrategia codiciosa sería elegir la caja más grande que quepa en el espacio restante y repetir hasta que no puedas empacar más cajas. No, esto no le dará la solución óptima.

**10.2** You're going to Europe, and you have seven days to see everything you can. You assign a point value to each item (how much you want to see it) and estimate how long it takes. How can you maximize the point total (seeing all the things you really want to see) during your stay? Come up with a greedy strategy. Will that give you the optimal solution?

10.2 Vas a Europa y tienes siete días para ver todo lo que puedes. Asignas un valor en puntos a cada elemento (cuánto quieres verlo) y estimas cuánto tiempo lleva. ¿Cómo puedes maximizar el total de puntos (ver todas las cosas que realmente quieres ver) durante tu estadía? Piensa en una estrategia codiciosa. ¿Eso le dará la solución óptima?

*Answer:* Keep picking the activity with the highest point value that you can still do in the time you have left. Stop when you can't do anything else. No, this won't give you the optimal solution.

Respuesta: Sigue eligiendo la actividad con el valor de puntos más alto que aún puedes realizar en el tiempo que te queda. Detente cuando no puedes hacer nada más. No, esto no le dará la solución óptima.

## CHAPTER 11

### CAPÍTULO 11

**11.1** Suppose you can steal another item: a mechanical keyboard. It weighs 1 lb and is worth \$1,000. Should you steal it?

11.1 Supongamos que puedes robar otro artículo: un teclado mecánico. Pesa 1 libra y vale \$1,000. ¿Deberías robarlo?

*Answer:* Yes. Then you could steal the mechanical keyboard, the iPhone, and the guitar, worth a total of

\$4,500.

Respuesta: Sí. Luego podrías robar el teclado mecánico, el iPhone y la guitarra, por un valor total de 4.500 dólares.

**11.2** Suppose you're going camping. You have a knapsack that holds 6 lb, and you can take the following items. They each have a value, and the higher the value, the more important the item is:

11.2 Supongamos que vas a acampar. Tienes una mochila con capacidad para 6 libras y puedes llevar los siguientes artículos. Cada uno tiene un valor y cuanto mayor sea el valor, más importante será el artículo:

- Water, 3 lb, 10  
Agua, 3 libras, 10
- Book, 1 lb, 3  
Libro, 1 libra, 3
- Food, 2 lb, 9  
Comida, 2 libras, 9
- Jacket, 2 lb, 5  
Chaqueta, 2 libras, 5
- Camera, 1 lb, 6  
Cámara, 1 libra, 6

What's the optimal set of items to take on your camping trip?

¿Cuál es el conjunto óptimo de artículos para llevar en su viaje de campamento?

*Answer:* You should take water, food, and a camera.

Respuesta: Debes llevar agua, comida y una cámara.

**11.3** Draw and fill in the grid to calculate the longest common substring between *blue* and *clues*.

11.3 Dibuje y complete la cuadrícula para calcular la subcadena común más larga entre el azul y las pistas.

*Answer:*

Respuesta:

	C	L	U	E	S
B	0	0	0	0	0
L	0	1	0	0	0
U	0	0	2	0	0
E	0	0	0	3	0

## CHAPTER 12

### CAPÍTULO 12

**12.1** In the Netflix example, you calculated the distance between two different users using the distance formula. But not all users rate movies the same way. Suppose you have two users, Yogi and Pinky, who have the same taste in movies. But Yogi rates any movie he likes as a 5, whereas Pinky is choosier and reserves the 5s for only the best. They're well matched, but according to the distance algorithm, they aren't neighbors. How would you take their different rating strategies into account?

12.1 En el ejemplo de Netflix, calculó la distancia entre dos usuarios diferentes usando la fórmula de distancia. Pero no todos los usuarios valoran las películas de la misma manera. Supongamos que tienes dos usuarios, Yogi y Pinky, que tienen el mismo gusto cinematográfico. Pero Yogi califica cualquier película que le guste con un 5, mientras

que Pinky es más exigente y reserva el 5 sólo para las mejores. Están bien emparejados, pero según el algoritmo de distancia, no son vecinos. ¿Cómo tomaría en cuenta sus diferentes estrategias de calificación?

*Answer:* You could use something called normalization. You look at the average rating for each person and use it to scale their ratings. For example, you might notice that Pinky's average rating is 3, whereas Yogi's average rating is 3.5. So you bump up Pinky's ratings a little until her average rating is 3.5 as well. Then you can compare their ratings on the same scale.

Respuesta: Podrías usar algo llamado normalización. Observa la calificación promedio de cada persona y la usa para escalar sus calificaciones. Por ejemplo, puedes notar que la calificación promedio de Pinky es 3, mientras que la calificación promedio de Yogi es 3,5. Así que aumentas un poco las calificaciones de Pinky hasta que su calificación promedio también sea 3,5. Luego podrá comparar sus calificaciones en la misma escala.

**12.2** Suppose Netflix nominates a group of "influencers." For example, Quentin Tarantino and Wes Anderson are influencers on Netflix, so their ratings count for more than a normal user's. How would you change the recommendations system so it's biased toward the ratings of influencers?

**12.2** Supongamos que Netflix nomina a un grupo de "influencers". Por ejemplo, Quentin Tarantino y Wes Anderson son personas influyentes en Netflix, por lo que sus calificaciones cuentan más que las de un usuario normal. ¿Cómo cambiarías el sistema de recomendaciones para que esté sesgado hacia las calificaciones de los influencers?

*Answer:* You could give more weight to the ratings of the influencers when using KNN. Suppose you have three neighbors: Joe, Dave, and Wes Anderson (an influencer). They rated Caddyshack 3, 4, and 5, respectively. Instead of just taking the average of their ratings ( $3 + 4 + 5 / 3 = 4$  stars), you could give Wes Anderson's rating more weight:  $3 + 4 + 5 + 5 + 5 / 5 = 4.4$  stars.

Respuesta: Podrías dar más peso a las calificaciones de los influencers al usar KNN. Suponga que tiene tres vecinos: Joe, Dave y Wes Anderson (un influencer). Calificaron a Caddyshack con 3, 4 y 5, respectivamente. En lugar de simplemente tomar el promedio de sus calificaciones ( $3 + 4 + 5 / 3 = 4$  estrellas), podría darle más peso a la calificación de Wes Anderson:  $3 + 4 + 5 + 5 + 5 / 5 = 4,4$  estrellas.

**12.3** Netflix has millions of users. The earlier example looked at the five closest neighbors for building the recommendations system. Is this too low? Too high?

12.3 Netflix tiene millones de usuarios. El ejemplo anterior analizó a los cinco vecinos más cercanos para construir el sistema de recomendaciones. ¿Es esto demasiado bajo? ¿Demasiado alto?

*Answer:* It's too low. If you look at fewer neighbors, there's a bigger chance that the results will be skewed. A good rule of thumb is that if you have  $N$  users, you should look at  $\sqrt{N}$  neighbors.

Respuesta: Es demasiado bajo. Si se analizan menos vecinos, existe una mayor probabilidad de que los resultados estén sesgados. Una buena regla general es que si tiene  $N$  usuarios, debe buscar los vecinos  $\sqrt{N}$ .

# **index**

---

## **índice**

---

### **A**

### **A**

algorithms  
algoritmos  
running times of, growth rates of [11–13](#)  
tiempos de ejecución de, tasas de crecimiento de [11 a 13](#)  
run-time analysis, worst-case run time [15](#)  
análisis de tiempo de ejecución, tiempo de ejecución en el peor de los casos [15](#)  
approximate solution [205](#)  
solución aproximada [205](#)  
approximation algorithms [197](#)  
algoritmos de aproximación [197](#)  
arrays  
matrices  
linked lists and [24–25](#)  
listas enlazadas y [24–25](#)  
terminology [27](#)  
terminología [27](#)  
ASCII (American Standard Code for Information Interchange) [134](#)  
ASCII (Código estándar americano para el intercambio de información) [134](#)  
asymmetric key encryption [259](#)  
cifrado de clave asimétrica [259](#)  
average case [70](#)  
caso promedio [70](#)  
quicksort, vs. worst case [75](#)  
clasificación rápida, frente al peor de los casos [75](#)  
AVL trees [151](#)  
árboles AVL [151](#)

## B

### B

balanced trees [141, 150, 164](#)  
árboles equilibrados [141, 150, 164](#)  
AVL trees [151](#)  
árboles AVL [151](#)  
B-trees [162–164](#)  
Árboles B [162–164](#)  
improving insertion speed with [143](#)  
mejorando la velocidad de inserción con [143](#)  
rotation [151](#)  
rotación [151](#)  
splay trees [159](#)  
árboles extendidos [159](#)  
balance factor [154](#)  
factor de equilibrio [154](#)  
base case, in recursion [44–45](#)  
caso base, en recursividad [44–45](#)  
Bellman-Ford algorithm [180](#)  
Algoritmo de Bellman-Ford [180](#)  
BFS (breadth-first search) [101, 120](#)  
BFS (búsqueda primero en amplitud) [101, 120](#)  
graphs, overview of [102–104](#)  
gráficos, descripción general de [102–104](#)  
implementing BFS algorithm [113–117](#)  
implementación del algoritmo BFS [113–117](#)  
implementing graph [111–112](#)  
implementar el gráfico [111–112](#)  
overview of [106, 109](#)  
descripción general de [106, 109](#)  
shortest path problem, using breadth-first search [108–109](#)  
problema del camino más corto, utilizando la búsqueda en amplitud [108–109](#)  
Big O notation [10, 75](#)  
Notación O grande [10, 75](#)  
average case vs. worst case [72–75](#)  
Caso promedio vs. peor caso [72–75](#)  
establishing worst-case run time [15](#)  
establecer el peor tiempo de ejecución [15](#)

merge sort vs. quicksort [71–72](#)  
clasificación por combinación frente a clasificación rápida [71–72](#)  
run times [15–17, 70](#)  
tiempos de ejecución [15–17, 70](#)  
traveling salesperson problem [17–19](#)  
problema del vendedor ambulante [17–19](#)  
visualizing run times [13](#)  
visualizando tiempos de ejecución [13](#)  
binary search [3, 142](#)  
búsqueda binaria [3, 142](#)  
running time of [10](#)  
tiempo de ejecución de [10](#)  
binary\_search function [8](#)  
función de búsqueda binaria [8](#)  
binary search trees (BSTs) [143](#)  
árboles de búsqueda binaria (BST) [143](#)  
binary trees [131, 139](#)  
árboles binarios [131, 139](#)  
Bloom filters [253](#)  
Filtros de floración [253](#)  
breadth-first search. See BFS (breadth-first search)  
búsqueda en amplitud. Ver BFS (búsqueda primero en amplitud)  
brute-force algorithms, knapsack problem [204](#)  
algoritmos de fuerza bruta, problema de mochila [204](#)  
BSTs. See binary search trees  
BST. Ver árboles de búsqueda binaria  
B-trees [161](#)  
árboles B [161](#)  
advantages of [162–164](#)  
ventajas de [162–164](#)  
bye function [48](#)  
función de despedida [48](#)

## C

## C

cache, using hash tables as [88](#)  
caché, usando tablas hash como [88](#)

call stack [46, 47–49, 49](#)  
pila de llamadas [46, 47–49, 49](#)  
character encoding [134](#)  
codificación de caracteres [134](#)  
collisions [91–93](#)  
colisiones [91–93](#)  
connected, acyclic graph [130](#)  
gráfico conectado y acíclico [130](#)  
constant [71](#)  
constante [71](#)  
constant time [94](#)  
tiempo constante [94](#)  
countdown example, base case and recursive case [44–45](#)  
ejemplo de cuenta regresiva, caso base y caso recursivo [44–45](#)

## D

### D

data structures  
estructuras de datos  
arrays  
matrices  
linked lists and [24–25](#)  
listas enlazadas y [24–25](#)  
overview of [26](#)  
descripción general de [26](#)  
selection sort and [24](#)  
clasificación de selección y [24](#)  
terminology [27](#)  
terminología [27](#)  
AVL trees [151](#)  
árboles AVL [151](#)  
balanced trees [147–150, 164](#)  
árboles equilibrados [147–150, 164](#)  
binary search trees (BSTs) [143](#)  
árboles de búsqueda binaria (BST) [143](#)  
binary trees [131, 139](#)  
árboles binarios [131, 139](#)

B-trees [161](#)  
árboles B [161](#)  
graphs  
graficos  
    BFS, implementing [111–112](#)  
    BFS, implementación [111–112](#)  
    Dijkstra's algorithm, implementation of [181–189](#)  
    Algoritmo de Dijkstra, implementación de [181–189](#)  
    overview of [102–105](#), [122](#)  
    descripción general de [102–105](#), [122](#)  
    terminology [170–172](#)  
    terminología [170–172](#)  
    weighted [165](#)  
    ponderado [165](#)  
data structures (*continued*)  
estructuras de datos (continuación)  
hash tables [77](#), [84](#), [91](#), [99](#), [111](#)  
tablas hash [77](#), [84](#), [91](#), [99](#), [111](#)  
heaps [261–263](#)  
montones [261–263](#)  
linked lists [24](#)  
listas enlazadas [24](#)  
probabilistic algorithms, HyperLogLog [255](#)  
algoritmos probabilísticos, HyperLogLog [255](#)  
queues, overview of [109](#)  
colas, descripción general de [109](#)  
recommendation systems [231](#)  
sistemas de recomendación [231](#)  
splay trees [159](#)  
árboles extendidos [159](#)  
stacks  
pilas  
    call stack [47–49](#)  
    pila de llamadas [47–49](#)  
    call stack with recursion [49–53](#)  
    pila de llamadas con recursividad [49–53](#)  
trees, depth-first search [126–129](#)  
árboles, búsqueda en profundidad [126–129](#)  
D&C. See divide and conquer (D&C)  
CORRIENTE CONTINUA. Ver divide y vencerás (D&C)  
decision problems [268](#)

problemas de decisión 268  
deletions 30  
eliminaciones 30  
depth-first search (DFS) 126–129  
búsqueda en profundidad (DFS) 126–129  
deque function 113  
función deque 113  
dictionaries 83  
diccionarios 83  
diff command 227  
comando diferencial 227  
Diffie-Hellman key exchange 255  
Intercambio de claves Diffie-Hellman 255  
ephemeral Diffie-Hellman key exchange 259  
intercambio efímero de claves Diffie-Hellman 259  
Dijkstra's algorithm 165, 190  
Algoritmo de Dijkstra 165, 190  
implementation of 181–189  
implementación de 181–189  
terminology 170–172  
terminología 170–172  
working with 166–170  
trabajando con 166–170  
directed graph 112  
gráfico dirigido 112  
distributed algorithms 252  
algoritmos distribuidos 252  
divide and conquer (D&C) 56–63  
divide y vencerás (DyC) 56–63  
quicksort 64–76  
clasificación rápida 64–76  
DNS cache 86  
Caché DNS 86  
DNS resolution 86  
Resolución DNS 86  
duplicates, preventing 86–88  
duplicados, evitando 86–88  
dynamic programming 228  
programación dinámica 228  
knapsack problem 203, 216  
problema de mochila 203, 216

**brute-force solution** 204  
**solución de fuerza bruta** 204  
**dynamic programming solution for** 205  
**solución de programación dinámica para** 205  
**FAQ** 213, 216, 217  
Preguntas frecuentes 213, 216, 217  
**filling in grid** 216  
rellenando la cuadrícula 216  
**filling knapsack completely** 220  
llenar la mochila completamente 220  
**items that depend on each other** 219  
elementos que dependen unos de otros 219  
**overview of** 206–208  
descripción general de 206–208  
**stealing fractions of items** 217  
robar fracciones de artículos 217  
**longest common substring problem** 220  
problema de subcadena común más largo 220  
    **filling in grid** 222–223  
    llenando la cuadrícula 222–223  
**longest common subsequence** 225  
subsecuencia común más larga 225  
**solution** 224–225  
solución 224–225

## E

### mi

**edges, negative-weight** 178–180  
**bordes, peso negativo** 178–180  
**enqueue** 113  
poner en cola 113  
**Euclid's algorithm** 58  
Algoritmo de Euclides 58

## F

## F

factorial function 49  
función factorial 49  
feature extraction 233–237, 243, 246  
extracción de características 233–237, 243, 246  
Feynman algorithm 222  
Algoritmo de Feynman 222  
FIFO (first in, first out) 110, 263  
FIFO (primero en entrar, primero en salir) 110, 263  
file directories 123–125  
directorios de archivos 123–125  
find\_lowest\_cost\_node algorithm 184  
algoritmo find\_lowest\_cost\_node 184  
Fourier transform 250  
Transformada de Fourier 250  
fruit classification example 229  
ejemplo de clasificación de frutas 229

## G

### GRAMO

golden ratio 266  
proporción áurea 266  
graphs  
graficos  
BFS, implementing 111–112  
BFS, implementación 111–112  
Dijkstra's algorithm, implementation of 181–189  
Algoritmo de Dijkstra, implementación de 181–189  
overview of 102–105, 122  
descripción general de 102–105, 122  
terminology 170–172  
terminología 170–172  
weighted 165  
ponderado 165  
greedy algorithms 191, 202  
algoritmos codiciosos 191, 202

knapsack problem [194–196](#)  
problema de mochila [194–196](#)  
set-covering problem [196–197, 201–202](#)  
problema de cobertura de conjuntos [196–197, 201–202](#)  
    calculating answer [199](#)  
    calculando la respuesta [199](#)  
    code for setup [198](#)  
    código para la configuración [198](#)  
    sets [201](#)  
        conjuntos [201](#)  
greet2 function [48](#)  
función saludar2 [48](#)  
greet function [48, 49](#)  
función de saludo [48, 49](#)  
growth rates of algorithms [11–13](#)  
tasas de crecimiento de los algoritmos [11–13](#)  
guitar row [206–208](#)  
fila de guitarra [206–208](#)

## H

### h

hash functions [97](#)  
funciones hash [97](#)  
hash tables [77, 84, 91, 99, 111](#)  
tablas hash [77, 84, 91, 99, 111](#)  
collisions [91–93](#)  
colisiones [91–93](#)  
lookups [84–86](#)  
búsquedas [84–86](#)  
performance [93–95](#)  
rendimiento [93–95](#)  
    hash functions [97](#)  
    funciones hash [97](#)  
    load factor [96–97](#)  
    factor de carga [96–97](#)  
preventing duplicate entries [86–88](#)  
prevenir entradas duplicadas [86–88](#)

**use cases, lookups** 84–86  
**casos de uso, búsquedas** 84–86  
**using as cache** 88  
**usando como caché** 88  
**heapsort algorithm** 262  
**algoritmo de clasificación de montón** 262  
**Huffman coding** 136, 133–139  
**Codificación de Huffman** 136, 133-139  
**HyperLogLog** 253, 255  
**HiperLogLog** 253, 255

## I

## I

**inductive proofs** 69  
**pruebas inductivas** 69  
**injective function** 83  
**función inyectiva** 83  
**in-neighbors** 105  
**vecinos** 105  
**ISO-8859-1 code** 133  
**Código ISO-8859-1** 133

## K

## k

**knapsack problem** 194–196, 203  
**problema de mochila** 194–196, 203  
**adding items to** 213–215  
**agregar elementos a** 213–215  
**adding smaller item** 216  
**añadiendo un elemento más pequeño** 216  
**changing order of rows** 216  
**cambiando el orden de las filas** 216  
**dynamic programming solution for** 205

solución de programación dinámica para 205

FAQ [213](#)

Preguntas frecuentes [213](#)

filling in grid row-wise [216](#)

rellenando la cuadrícula por filas [216](#)

filling knapsack completely [220](#)

llenar la mochila completamente [220](#)

items that depend on each other [219](#)

elementos que dependen unos de otros [219](#)

optimizing travel itinerary [217](#)

optimización del itinerario de viaje [217](#)

overview of dynamic programming [206–208](#)

descripción general de la programación dinámica [206–208](#)

revisited, brute-force solution [204](#)

solución revisada de fuerza bruta [204](#)

stealing fractions of items [217](#)

robar fracciones de artículos [217](#)

k-nearest neighbors (KNN) [229](#)

k-vecinos más cercanos (KNN) [229](#)

building spam filters, Naive Bayes classifier [243](#)

creación de filtros de spam, clasificador Naive Bayes [243](#)

overview of machine learning [241](#)

descripción general del aprendizaje automático [241](#)

recommendation system

sistema de recomendación

feature extraction [233–237](#)

extracción de características [233–237](#)

features for [240](#)

características para [240](#)

regression [238–240](#)

regresión [238–240](#)

stock market, predicting [244](#)

mercado de valores, prediciendo [244](#)

training models, high-level overview of [224–228, 229–246](#)

modelos de capacitación, descripción general de alto nivel de [224–228, 229–246](#)

L

I

Levenshtein distance 227  
distancia de Levenshtein 227  
LIFO (last in, first out) 110, 263  
LIFO (último en entrar, primero en salir) 110, 263  
linear programming 263  
programación lineal 263  
linear regression 247  
regresión lineal 247  
linear time 15  
tiempo lineal 15  
linked lists 24  
listas enlazadas 24  
deletions 30  
eliminaciones 30  
selection sort 25, 31–34  
clasificación de selección 25, 31–34  
terminology 27  
terminología 27  
load factor 96–97  
factor de carga 96–97  
logarithmic time. See log time  
tiempo logarítmico. Ver tiempo de registro  
logarithms 7  
logaritmos 7  
log time 10, 15  
tiempo de registro 10, 15  
longest common substring problem 220  
problema de subcadena común más largo 220  
filling in grid 222–223  
llenando la cuadrícula 222–223  
longest common subsequence 225  
subsecuencia común más larga 225  
making grid 221–222  
haciendo la cuadrícula 221–222  
solution 224–225  
solución 224–225  
LSH (locality-sensitive hashing) 260–261  
LSH (hash sensible a la localidad) 260–261

## METRO

machine learning, overview of 241  
aprendizaje automático, descripción general de 241  
map function 252  
función de mapa 252  
map/reduce 252  
mapa/reducir 252  
merge sort 70  
fusionar ordenar 70  
vs. quicksort 71–72  
frente a clasificación rápida 71–72  
min heaps 261–263  
montones mínimos 261–263  
ML (machine learning)  
ML (aprendizaje automático)  
stock market, predicting 244  
mercado de valores, prediciendo 244  
training models, high-level overview of 244–246  
modelos de entrenamiento, descripción general de alto nivel de 244–246

## N

### norte

Naive Bayes classifier 243  
Clasificador ingenuo de Bayes 243  
negative-weight edges 178–180  
bordes de peso negativo 178–180  
nondeterministic polynomial time (NP) 267  
tiempo polinómico no determinista (NP) 267  
NP-complete problems 267, 268, 275–276  
Problemas NP-completos 267, 268, 275–276  
NP-hard problems 267, 274–275  
Problemas NP difíciles 267, 274–275  
hard to solve, quick to verify 272  
difícil de resolver, rápido de verificar 272  
overview of 274–275  
descripción general de 274–275

reductions 273  
reducciones 273  
satisfiability problem 269–271  
problema de satisfacibilidad 269–271  
NP (nondeterministic polynomial time) 267  
NP (tiempo polinómico no determinista) 267

## O

### oh

optical character recognition (OCR) 242  
reconocimiento óptico de caracteres (OCR) 242  
out-neighbors 105  
vecinos 105

## P

### PAG

parallel algorithms 251  
algoritmos paralelos 251  
partitioning 65  
partición 65  
perfect hash function 83  
función hash perfecta 83  
performance 2  
rendimiento 2  
pivot 64  
pivote 64  
pointers 30  
punteros 30  
pop 113  
pop 113  
power set 196  
conjunto de potencia 196  
print function 47

función de impresión 47  
priority queues [261–263](#)  
colas de prioridad 261–263  
probabilistic algorithms, HyperLogLog [255](#)  
algoritmos probabilísticos, HyperLogLog 255  
probabilistic data structures [254](#)  
estructuras de datos probabilísticos 254  
push [113](#)  
empujar [113](#)  
P vs. NP problem [273](#)  
Problema P versus NP 273

## Q

### q

queue data structure [109](#)  
estructura de datos de cola 109  
quicksort [64–76](#)  
clasificación rápida 64–76  
average case vs. worst case [72–75](#)  
Caso promedio vs. peor caso 72–75  
Big O notation [75](#)  
Notación O grande 75  
divide and conquer [56–63](#)  
divide y vencerás 56–63  
merge sort vs. [71–72](#)  
fusionar ordenación vs. 71–72  
run times for [70](#)  
tiempos de ejecución para 70

## R

### R

*< i>Real-World Cryptography</i>* (Wong) [260](#)  
*< i>Criptografía del mundo real</i>* (Wong) 260

recommendation systems [231](#)  
sistemas de recomendación [231](#)  
feature extraction [233–237](#)  
extracción de características [233–237](#)  
recursion [41, 44, 54](#)  
recursión [41, 44, 54](#)  
base case and recursive case [44](#)  
caso base y caso recursivo [44](#)  
call stack [47–49](#)  
pila de llamadas [47–49](#)  
overview of [42](#)  
descripción general de [42](#)  
stacks, call stack with recursion [49–53](#)  
pilas, pila de llamadas con recursividad [49–53](#)  
reduce function [252](#)  
reducir la función [252](#)  
reductions [273](#)  
reducciones [273](#)  
regression [239](#)  
regresión [239](#)  
regression analysis, linear [247](#)  
análisis de regresión, lineal [247](#)  
regression, KNN (k-nearest neighbors) [238–240](#)  
regresión, KNN (k-vecinos más cercanos) [238–240](#)  
rotation, in AVL trees [151, 154](#)  
rotación, en árboles AVL [151, 154](#)  
run-time analysis, worst-case run time [15](#)  
análisis de tiempo de ejecución, tiempo de ejecución en el peor de los casos [15](#)  
run times [15–17](#)  
tiempos de ejecución [15–17](#)  
visualizing [13](#)  
visualizando [13](#)

## S

## S

SAT (satisfiability) problem [269–271](#)  
Problema SAT (satisfacibilidad) [269–271](#)

search, binary search [5–9](#)  
búsqueda, búsqueda binaria [5–9](#)  
selection sort [21, 34, 38, 39](#)  
selección tipo [21, 34, 38, 39](#)  
arrays and linked lists [24, 31](#)  
matrices y listas enlazadas [24, 31](#)  
    terminology [27](#)  
    terminología 27  
arrays, overview of [26](#)  
matrices, descripción general de [26](#)  
deletions [30](#)  
eliminaciones 30  
insertion sort, inserting into middle of list [29](#)  
ordenación por inserción, insertando en el medio de la lista [29](#)  
memory [22–23](#)  
memoria 22-23  
selection sort, code listing [38](#)  
clasificación por selección, listado de códigos 38  
set-covering problem [196, 267](#)  
problema de cobertura de conjuntos [196, 267](#)  
calculating answer [199](#)  
calculando la respuesta 199  
code for setup [198](#)  
código para la configuración 198  
greedy algorithms for  
algoritmos codiciosos para  
    approximation algorithms [197](#)  
    algoritmos de aproximación 197  
shortest path [178](#)  
camino más corto 178  
shortest-path algorithms, Dijkstra's algorithm [165](#)  
algoritmos del camino más corto, algoritmo de Dijkstra 165  
shortest-path problem [104](#)  
problema del camino más corto 104  
terminology [170–172](#)  
terminología 170–172  
using breadth-first search [108–109](#)  
utilizando la búsqueda en amplitud [108–109](#)  
simple search [5](#)  
búsqueda sencilla 5  
spam filters, building with KNN [243](#)

filtros de spam, construyendo con KNN 243  
splay trees 159  
árboles extendidos 159  
stack data structure 46  
estructura de datos de pila 46  
stacks  
pilas  
call stack 47–49  
pila de llamadas 47–49  
call stack with recursion 49–53  
pila de llamadas con recursividad 49–53  
stat command 133  
comando de estadística 133  
subproblem 215  
subproblema 215  
sum function 61, 63  
función de suma 61, 63  
symmetric key encryption 259  
cifrado de clave simétrica 259

## T

### t

tail recursion 53  
recursión de cola 53  
topological sort 118  
clasificación topológica 118  
training 243  
entrenamiento 243  
traveling salesperson problem 17–19, 267  
problema del vendedor ambulante 17-19, 267  
travel itinerary, optimizing 217  
itinerario de viaje, optimizando 217  
tree algorithms, file directories 123–125  
algoritmos de árbol, directorios de archivos 123–125  
tree insertion, improving speed with 143  
inserción de árboles, mejorando la velocidad con 143  
trees 119, 121, 140

árboles 119, 121, 140  
depth-first search 126–129  
búsqueda en profundidad 126–129  
Huffman coding 133–139  
Codificación de Huffman 133-139  
overview of 122  
descripción general de 122

## U

### Ud.

unweighted graphs 170  
gráficos no ponderados 170  
UTF-8 (Unicode Transformation Format 8) 134  
UTF-8 (formato de transformación Unicode 8) 134

## V

### V

vectors 236  
vectores 236

## W

### W.

weighted graphs 170  
gráficos ponderados 170  
weights 170  
pesos 170  
Wong, David, *< i>Real-World Cryptography</i>* 260  
Wong, David, *< i>Criptografía del mundo real</i>* 260  
worst case 70  
peor caso 70