Think in Recursion in Algorithmic Programming

Write recursive computer algorithms to solve complex problems

Dr. Fang Jin

www.bpbonline.com

First Edition 2025

ISBN: 978-93-65891-348

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

*To my wife, **Maggie**, for the support of the family and patience for the life drama.*

*To my two kids, **Sophia** and **Lucas**, for all the inspiration and curiosity.*

To my father and mother, who supported me throughout my whole life, especially during my difficult time.

Without you, all of this would not have been possible.

– Dr. Fang Jin

About the Author

Dr. Fang Jin is a software engineer who makes reusable and scalable web modules that can be applied to modern browsers and devices. He is an engineering hobbyist with interests ranging from economics and philosophy to software development, architecture, and team development. Fang is a lifelong learner, and throughout his journey, he has helped organizations and teams adopt UI frameworks to achieve consistent branding and look and feel across departments so that they can save costs and boost efficiency. In his free time, he likes swimming, running, and surfing YouTube.

Acknowledgement

I would like to express our sincere gratitude to all those who contributed to the completion of this book.

First and foremost, I extend my heartfelt appreciation to my family and friends for their unwavering support and encouragement throughout this journey. Their love and encouragement have been a constant source of motivation.

I am immensely grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. Their support and assistance were invaluable in navigating the complexities of the publishing process.

I would also like to acknowledge the reviewers, technical experts, and editors who provided valuable feedback and contributed to the refinement of this manuscript. Their insights and suggestions have significantly enhanced the quality of the book.

Last but not least, I want to express my gratitude to the readers who have shown interest in this book. Your support and encouragement have been deeply appreciated.

Thank you to everyone who has played a part in making this book a reality.

Preface

In the fifth grade, I had my first programming lessons in some outside-school activity. I still remember the teacher asking us to draw diamond shapes on the computer screen using shape characters such as stars, etc. To be honest, that was one of the best times of my life. Now I can see, why, after I finished my PhD in chemical engineering and a couple of years of cancer research as a postdoc, I decided to take an entry level job as a programmer working with computer languages. I guess my destiny has been determined by what was memorable in my very early years.

Over many years of working as a programmer, especially after practicing in the field for a couple of years, I started to benefit tremendously from two computer concepts other than computer algorithms in general. One is functional programming, and another is recursion. Both are things that I would never have thought I would learn, and both have changed my ways of writing code, or may have even inspired me how to live my life.

If there is one thing in common with what I learned from them, it is, an open mind. There are always multiple ways in life. Recursion is one of those ways. It is mysterious and might have an intimidating reputation.

This book is about to show you the different way. Instead of focusing on the difference between recursion and the normal way, the book wants to prove to you in the end that recursion is a more natural way of human reasoning, especially in approaching complex problem solving. In order to achieve that, the book will walk you through the problems of recursion after explaining to you the key elements of recursion. The problem is categorized mainly by the dimension as well as the main data structure used. The problems ranges from common daily activities, interview questions, or even some mindblogging competition problems. By the end of the book, you should be equipped with the right mindset to write a recursion algorithm from scratch by yourself. Whether you are a novice or an experienced learner, I hope this book will serve as a valuable resource in your journey of exploring the possibilities of using recursion as one of your main reasoning toolsets.

Chapter 1: Introduction– In this opening chapter, we will talk about what we need to have before reading this book. By the end of this chapter, you should be able to read code from this book and run each of them on the fly. You will also get a sneak peek at what the minimum features are required to support functional programming in general.

Chapter 2: What is Recursion? The chapter will define recursion in computer algorithms by writing down our first recursion algorithm step by step. By the end of this chapter, you

will be able to write the first recursion for the sum algorithm. You will also learn how to read and understand a recursion in general. After that, you will learn how to measure the recursion's running time using the big O notation. In the end, we will visit the printed results produced by the sum algorithm.

Chapter 3: Recursion as the Architect– This chapter will talk about the recursion's counterpart, iteration. We will put iteration and recursion side by side and study why we this book suggests you start think in recursion on top of the iteration. By the end of this chapter, you will understand in which area the iteration starts to run short in modeling complex problems and why recursion can serve as the architect for common problems, attributing to its flexibility of expressing complex structures freely. You will get to know when is the best time to learn and practice the recursion setup. At the end of the chapter, you will see how the recursion is run inside the computer compiler by using the function stack, how we can run into stack overflow issues with large amount of function calls, as well as how we suggest to deal with the stack overflow issue for recursion algorithms.

Chapter 4: Factorial and Power– This chapter asks you to apply what we have learned so far to two basic problems, giving you a chance to practice constructing the recursion from scratch. By the end of this chapter, you will write two recursion algorithms for the factorial number and the power of a base. For each, you will get to calculate its running times, support large integer number in the calculation, as well as running the algorithm avoiding the stack overflow limit. You will also get a detailed description of the inner-outer functions setup that we have been using so far to construct recursions. Last but not least, the printout of the tabular version of the factorial number, power of two, power of three and power of four are generated for quick reference by the rest of the book chapters.

Chapter 5: Fibonacci Sequence– This chapter asks you to write out the famous Fibonacci sequence using the recursion. The problem is a bit more complicated and would require us to apply a memoization technique to fully solve it in the end. In this chapter, you will be able to generate a Fibonacci sequence with the recursion. You will get a chance to understand why the sequence requires '2^n' computational power to accomplish. You will also learn how to apply memoization to reduce the computation to 'O(n)'. You will get to generate the Fibonacci numbers for larger 'n' by supporting big integers, as well as coming up with the linear recursion version to avoid the stack overflow issue. Finally, you will learn how to print out all Fibonacci numbers in a tabular format.

Chapter 6: Climbing Stairs– This chapter continues to see multiple invocation behavior of the recursive function. Moreover, the number of instances invoked can vary dynamically. In this chapter, you will get to solve the climbing stairs problem by formulating the recursion and constructing the boundary from all stair climbing scenarios. Later, we will write the

recursion algorithm, apply the memoization optimization, add support to the big number, and make sure we can climb as many stairs as possible. In the end, we will print out the climbed stairs sequence for maximum steps up to 2, 3, and 4 in tabular format.

Chapter 7: Edit Distance– This chapter presents another interesting problem that you might encounter at an interview or a competition. It is about changing a word into another word. In this chapter, you will learn how to solve the edit distance problem using recursion. You will get a chance to analyze this algorithm's running time. Afterward, we will apply the memoization optimization to speed up its running speed to 'O(k),' and in the end, we will further increase 'k' by using a linear recursion version that is derived from dynamic programming.

Chapter 8: Paint Bucket– This chapter will start to explore topics relating to directions in a two-dimensional space. This serves as a preliminary problem among the path-finding problems that we will solve in later chapters of this book. In this chapter, you will get to implement the famous Photoshop functionality paint bucket to fill an area using a recursion. We will walk through the detailed paint process to analyze the running time spent on the paint process. In the end, we will write a linear recursion version using a stack to avoid stack overflow issues so that we can use it to paint a large map size.

Chapter 9: Permutation– This chapter dives into the backtracking of recursion in detail. You will see it helps us set up an optimized, brutal force like infrastructure to build the solution along the way and regularly falls back to previous backtracks when the solution gets stalled. In this chapter, we will solve the permutation problem by selecting a number of items from a list of items and seeing how many ways we can do that. We will see how we can write code for a variable number of loops using the backtracking technique, especially how we manage backtracks within the recursion. We will calculate the algorithm's running time and come to know what the brutal force is. Afterward, we will compare back-to-back details of the backtracking and regular recursion. In the end, we will implement a linear recursion for permutation with a stack.

Chapter 10: Knapsack 0/1– This chapter will continue to use backtracking to solve another classical problem, the knapsack problem. In this chapter, we will solve the knapsack problem by asking how many items can fit into a bag to be the most valuable collection. We will first use the backtracking technique and figure out its running time. We will also get a chance to see how to do the mutable recursion version, and a linear version using a stack. We then revisit applying the regular recursion to solve the knapsack problem to see the difference. We will apply the memoization technique to it in two dimensions this time. In the end, we will solve the knapsack with a dynamic programming approach to achieve faster performance.

Chapter 11: Eight Queens– This chapter continues to apply the backtracking to a well-studied problem called the eight queens puzzle, where we place queens on a chess board and make sure they don't attack each other. In this chapter, we will solve the N-queens problem by placing 'n' number of queens on a chess board and making sure they do not attack each other horizontally, vertically, or diagonally. It is a fun problem, and we will use the backtracking technique to find all possible ways to place the queens. Afterwards, we will calculate the algorithm's running time and fine-tune the diagonal check to speed up the calculation. We also come up with an iterative solver for n-queen problem. In the end, we will list all possible ways of placing queens for various queen sizes.

Chapter 12: Finding Path– This chapter continues our journey of using recursion and backtracking. This time, we will apply them to another field: path-finding. In this chapter, we will start to find a path between two locations on a map using the backtracking approach. Then, we will study its running time and remove the stack overflow restriction by using a stack. We will point out that the path we found is not necessarily the shortest one and explain why it is not. In the end, we would like to attempt to find the shortest path by finding all possible paths between two points.

Chapter 13: Tree Traversal– This chapter will start to understand the tree traversal that drives the underlying visiting order of each function instance for the recursion. By following different visit orders, we can find the shortest path between two locations more efficiently. In this chapter, we will introduce two traversal orders for navigating nodes among a tree: the depth-first order and the breadth-first order. We will write an algorithm that gives us the depth-first order traversal, including two variations such as the preorder and the postorder. Then, we will understand how a queue can turn a depth-first order into a breadth-first order traversal. We will apply breadth-first order to our path-finding problem to find the shortest path. In the end, we make a brief comparison between the depth-first order and the breadth-first order.

Chapter 14: Shortest Route– This chapter navigates a map again and wants to collect items that we can find along the way, and in the end, make sure we take the shortest route of doing that. We will use the shortest distance algorithm we wrote in the previous chapter to come up with a map explorer to find all items. We will then apply the backtracking we learned to find all possible routes and select the shortest one among them. Next, we will figure out the running time of finding all routes and then apply a memoization scheme based on the past routes to speed up the calculation. Ultimately, we will explain the heuristics for finding a shorter path in general.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/j4nzj22

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Think-in-Recursion-in-Algorithmic-Programming.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction

Introduction

This book is all about recursion, an intuitive way of thinking. In this opening chapter, let us discuss what we need to know before reading it, such as how to read the code. We will also touch on functional programming, which is the basis of the book's code.

Structure

This chapter covers the following topics:

- Before starting recursion
- Making sense of code
- Functional programming
  - Function scope
  - Function as variable
  - Destructuring

Objectives

By the end of this chapter, you should be able to read code from this book and run each of them on the fly. You will also get a sneak peek at the minimum features required to support functional programming in general.

Before starting recursion

Recursion is a way of thinking. It is the skill of unfolding a complex structure into self-similar scopes. Each scope appears to play its role in an atomic fashion within an isolated space but manages to communicate in between and, in the end, allow us to answer questions on a larger scale. This is what makes recursion intriguing and powerful.

The human brain is more powerful than you think. The growth of the brain is mostly driven by needs. It can function the way you allow it to. So, you can train yourself in other ways of thinking if you choose to.

Have you wondered how some people turn out smarter than the rest? It is not normally believed that one person is born *smarter* than another. What really happens is how the brain develops after birth, especially what skills people develop at an early age.

The skill of thinking in recursion can be developed. Not only are we unlikely born to think that way, but it is one kind of logical reasoning that is developed based on the complexity we embrace as we interact with the world. This need to search for a qualified model to capture complex environments comes later in life.

This book explains how recursion plays a major part in human reasoning. It shows you how to write a recursion program through basic examples and how it can be applied to solve real problems. Along the way, we introduce more advanced techniques used in recursion so that we can use them to solve more difficult problems, either through brutal forces or their variations.

It will be quite a journey. By the time you finish half the book, you should be able to write a recursion from scratch by yourselves, and by the time you finish it, you should be able to think in recursion.

Making sense of code

The codes in this book are all written in JavaScript. They are published online in a readable and runnable notebook format on **observablehq.com**. The following link is the group link for all chapters:

https://observablehq.com/collection/@windmaomao/think-in-recursion/2

This modern online site displays the code along with comments written in Markdown, so you can read along while you run the code. The button to run is on the arrow key to the right of each code piece. In case you change any code, you can use the button to rerun it. Please refer to the following figure: