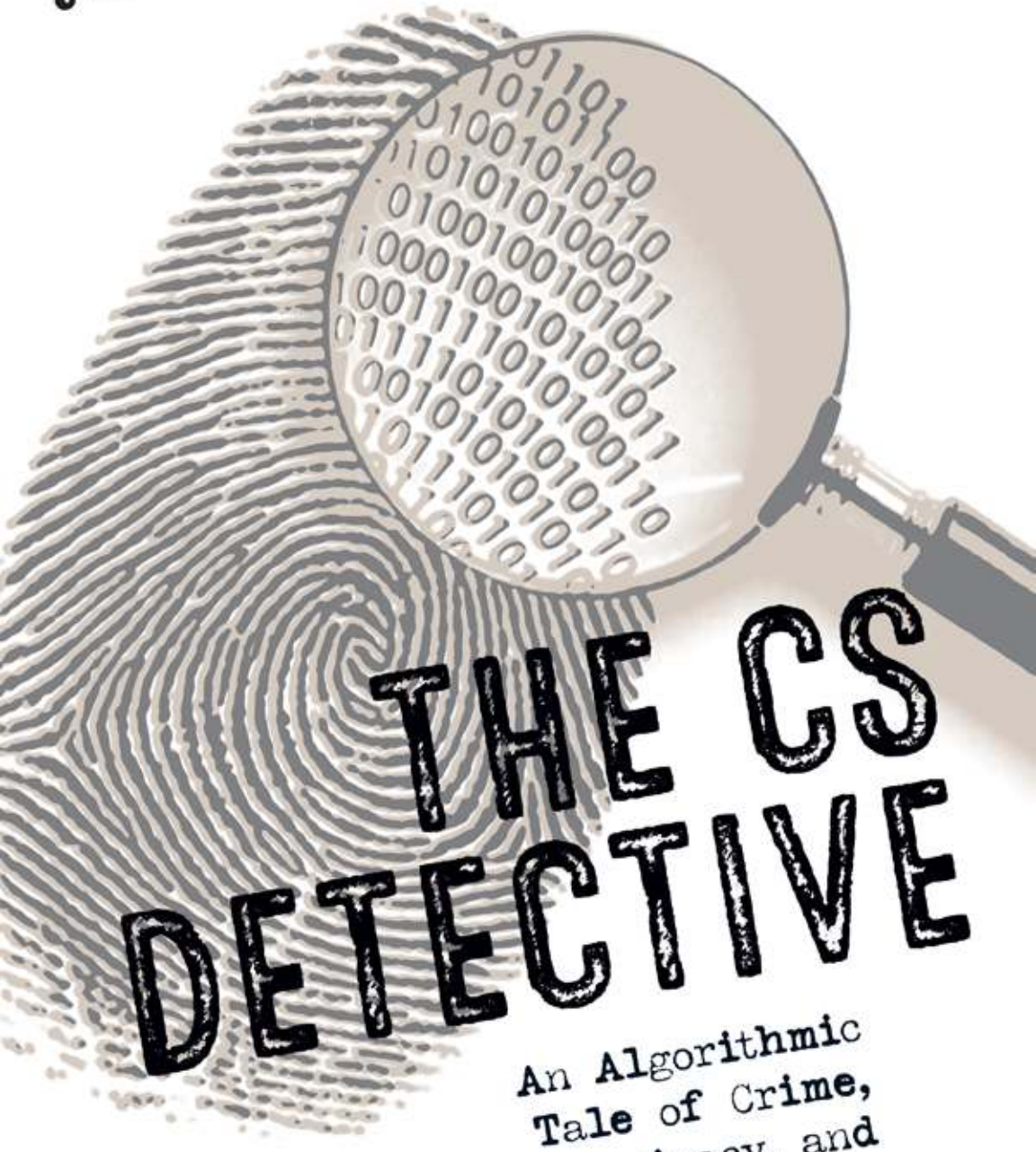
jeremy kubica



THE CS DETECTIVE

An Algorithmic Tale of Crime, Conspiracy, and Computation

Printed in USA

First printing

20 19 18 17 16 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-749-0
ISBN-13: 978-1-59327-749-9

Publisher: William Pollock
Production Editor: Riley Hoffman
Cover Design: Beth Middleworth
Illustrator: Miran Lipovača
Developmental Editor: Liz Chadwick
Technical Reviewer: Heidi Newton
Copyeditor: Rachel Monaghan
Compositor: Riley Hoffman
Proofreader: Paula L. Fleming

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
phone: 415.863.9900; info@nostarch.com
www.nostarch.com

Library of Congress Cataloging-in-Publication Data
A catalog record of this book is available from the Library of Congress.

Contents

A Note to Readers

This book focuses on computational thinking and search algorithms. The stories introduce and illustrate computational concepts at a high level, exploring the motivation behind them and their application in a noncomputer domain. This book is not a comprehensive text, and the stories are not intended as a substitute for a solid technical description of computer science. Instead, they are meant to be used like illustrations: they supplement the full concept and aid understanding.

The book covers a variety of computational approaches that all share the broad categorization of *search algorithms*. Concepts are presented first within the context of the story, and then explained more technically in a section laid out as lecture notes at the end of each chapter. Readers can safely skip these technical sections without missing any of the story.

This book assumes some experience with basic computer science concepts but does not require knowledge of any specific programming language. The algorithms in this book are meant to apply to a range of programming languages and problem domains.

— 1 —
Search Problems

The door opened without a knock—only the hinge's creak announced the visitor. Frank started for his crossbow, but pulled up short. If the Vinettees were coming for him, they would have knocked—with an axe. Whoever was coming through the door must want to talk. Frank reached for his mug instead and downed the remainder of his now-cold coffee.

"Captain Donovan," he said as the man entered. "What brings you to this fine neighborhood? I thought you didn't venture below Fifteenth Street anymore."

"It's been a while," the captain said simply. "How've you been, Frank?"

"Spectacular," Frank answered dryly, eyeing the captain as he walked a slow circuit around the room.

Donovan scanned Frank's shabby office. His red officer's cloak swished gently behind him. "How's the private eye game?"

"It pays the bills," Frank lied.

The captain nodded. He paused for a moment, then moved to the bookshelf and browsed the contents.

"So is this a social visit then?" Frank said. "Should I be asking after Marlene and the kids?"

"They're quite well," replied Donovan without turning around. "Marlene's turtle-grooming business is doing well these days. Bill joined the force last year. And Veronica is an accountant, just about the last thing we would have—"

"I wasn't actually asking," Frank interrupted.

The captain shrugged. He pulled a book from the shelf and leafed through the pages. Frank craned his neck to see the cover—*Police Academy Yearbook: Class XXI*.

"What do you want, Captain?" Frank demanded.

The captain met Frank's stare at last. "I need your help, Frank," he said.

Frank straightened. In the five years since Frank had left the force, the captain had paid him exactly two visits, and both had been to warn him to stay away from active cases. Threats were all Frank had come to expect, but now it seemed the captain had a special kind of problem—perhaps the kind that would mean an end to Frank's delinquent rent.

"I'm not on the force anymore," said Frank airily. "Why don't you get one of your trusted detectives to do it?"

"I need someone outside of the force," said the captain. "Drop the act, Frank. If you don't know what it means for me to be here, you're not the person I need."

Frank chuckled. "A leak? On *your* force?"

"Worse. Last night someone broke into the station's record room and stole over 500 scrolls."

"What were they after?" asked Frank. Without thinking, he leaned forward in his chair and reached for a fresh scroll and a quill. The movement came automatically to him, like drinking coffee or avoiding stairs.

"I don't know," said Donovan. "There was no pattern. They stole whole shelves of documents, everything from property disputes to expense reports. They took all the ledgers we keep on assassins, celebrities, private investigators, notaries . . . They even took both boxes of Farmer Swinson's noise complaints. But other shelves were completely untouched. We counted at least 512 missing documents."

"Maybe it was one of Farmer Swinson's neighbors," joked Frank. "They must've heard that after a mere hundred complaints, an intern will come to your house and give you a stern lecture."

Captain Donovan didn't bother to reply. He just stared pityingly until Frank cleared his throat and broke the silence. "So you want me to find these documents?"

The captain shook his head. "I want you to find the thieves. We have backups of the documents. I want to know what information they needed and what they plan to do with it."

"A search problem," Frank mused. During his time on the force, his two specialties had been search problems and annoying the captain.

"Does the king know?" Frank asked.

"I briefed him yesterday," said the captain, a hint of annoyance in his voice. "Ever since the trouble with that crackpot wizard, the king insists on daily briefings on everything." Two years ago, a megalomaniac wizard named Exponentious had tried to destroy the entire kingdom. Since then King Fredrick had personally instituted

sweeping upgrades to the kingdom's security, with over 300 new security regulations, at least 5 of which dealt with the storage of official documents in government buildings under 10 stories tall.

"I can't blame him though," Donovan grumbled. "It was a close call. If it hadn't been for Princess Ann, who knows where the kingdom would be now."

Frank nodded silently. Exponentious had attacked the algorithmic foundations of the kingdom by cursing the scholars who studied those algorithms. Within months he had rendered even simple operations inefficient, and the kingdom had started to grind to a halt. Evidence of the damage had been everywhere; even in his local bakery, Frank had himself witnessed panic break out as customers discovered they couldn't remember how to arrange themselves into a line.

"The king has, of course, taken a personal interest in the matter," the captain continued irritably. "He wants all the details: Who's assigned to the case? Which search algorithms are we using? Have we scoured all of the neighboring buildings?"

Frank stifled a chuckle and mulled over the proposition. A consulting gig for the capital's police force would be good money. He glanced down at his feet, where the tip of a toe peeked through a hole in his shoe. "If I'm going to consult," he said, "I'm going to do things my way."

This was the moment of truth. Five years ago he'd been kicked off the force for *doing things his way*. The captain was a man of rules and order. Frank's last use of heuristics had been the final straw—Captain Donovan had claimed his badge that very afternoon. But, then again, doing things his own way had always gotten Frank results.

"I figured as much," the captain responded at last. He pulled a thin folder from under his trench cloak and dropped it on Frank's desk.

"I'll be in touch," Donovan said. Then, without ceremony, he turned and left the office.

Three hours and twelve mugs of coffee later, Frank sat hunched over his desk and thumbed through the thin folder of information for the seventh time. The words jumped and swayed in the flickering candlelight, but didn't provide any new insights.

There wasn't a lot to go on. The captain had given him a list of missing documents and the duty roster for the night in question, but nothing more.

Finally, with an exaggerated sigh, Frank grabbed a piece of parchment and started making notes.

The first step in any search problem is determining what it is you hope to find—the *target*, as his old instructor in Police Algorithms 101 called it. Frank had learned that lesson early; he'd been tasked in his first week as an officer with finding the duke's prize stallion, and he'd proudly returned to the station that same afternoon with a 42-pound horned turtle. Apparently, the impressive reptile wasn't good enough. A good search algorithm means nothing if you're looking for the wrong thing.

In this case it wasn't a *what*, but rather a *who*. The captain had been right about that point. Once the thieves had the documents, it didn't matter if the police got them back. The thieves already had whatever information they needed.

So his target was simple: the person or persons who stole the documents.

The second step in any search problem is identifying the *search space*. What are you searching? During Frank's daily search for his keys, the search space was every flat surface in his office. And when

Frank wanted to find a criminal, his search space was every person in the vicinity of the capital.

Frank sat back and rubbed his eyes. It was a big search problem, finding a specific criminal in a city of criminals. But he had seen worse.

Now that he had defined the problem, he could start on an algorithm. A linear search was out; he couldn't afford to question everyone in the city. He could also rule out many of the other, fancier algorithms he had studied in the academy. For a problem like this, he would have to go back to his toolkit of basic search algorithms—the private investigator's most trusted friends.

Frank made a note on the parchment. He had the target to find, he knew the search space, and he had his algorithm. It was time to get to work.

POLICE ALGORITHMS 101: SEARCH PROBLEMS
Excerpt from Professor Drecker's Lecture

In this class we'll discuss several different algorithms (and related data structures) for solving search problems. A *search problem* is defined as any problem that requires us to find a specific value (or target) within a space of possible values (a search space).

Those of you who graduate and go on to become police officers will find yourselves facing problems that fall into this category every single day. This broad definition of a search problem encompasses a lot of different computational problems, from searching the police log for a specific entry to finding rooms within a hideout to finding all arrest records that match some criteria. This class won't be exhaustive—that would take years—but I'll give you some simple examples of basic and important algorithms as we go.

The algorithms described in this class will have three common components:

Target The piece of data you're searching for. The target can be either a specific value or a criterion that signifies the successful completion of a search.

Search space The set of all possibilities to test for the target. For example, the search space could be a list of values or all the nodes in a graph. A single possibility within the search space is called a *state*.

Search algorithm The set of specific steps or instructions for conducting the search.

Some search problems will have additional requirements or complexities, which we'll touch upon as we go over different algorithms.

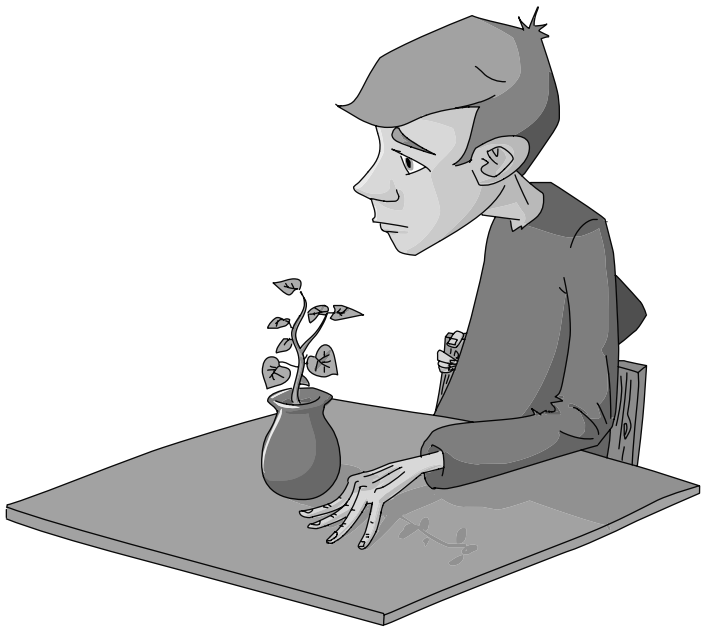
— 2 —
Exhaustive Search for an Informant

The key to efficient algorithms is information." It was Professor Drecker's mantra, barked at the cadets at the start of every Police Algorithms class, ferociously enough to sear itself permanently into Frank's memory. "A good algorithm depends on finding the structure in the data and using it. It depends on information."

Frank smiled to himself at the memory as he turned onto Three Bit Lane, a rutted dirt road lined with a combination of seedy bars and upscale coffee shops. He nodded politely to a pair of passing knights, who clanged as they jittered past in their armor, and made a mental note to grab a Triple-Bold Espresso before he left. First he needed information, something to help guide his search. He knew exactly where to start.

Glass Box Billy would be in one of the establishments by now, sitting quietly and listening to the wisps of conversation drifting through the room. People didn't mean to say things around Billy; they simply didn't notice he was there. Billy had been blessed with a single notable talent: utter inconspicuousness. Whatever he tried, there was something about Billy that meant people just didn't notice him. Maybe it was his pale skin or his small physique; maybe it was

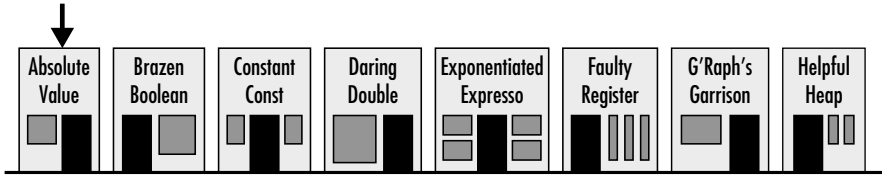
his exceptionally mundane taste in clothing. Whatever it was, Billy had long ago decided to put his one talent to use by eavesdropping, collecting information, and selling it to anyone who would buy.



Frank eyed the eight storefronts hunched together in Three Bit Lane and wondered which one Billy would have chosen. He ran through half a dozen search algorithms in his head, but it was pointless. Frank didn't have any information to go on. Billy could be in any one of the bars or coffee shops.

He'd have to use an exhaustive search—simply try all the possibilities until he found Billy. It didn't sit well with him. Years of the detective and private investigation game had taught him that there was almost always a better algorithm than exhaustive search, and he hated resorting to something so inefficient.

Grumbling, Frank started his search. He walked into the first bar on the street, The Absolute Value.



The bartender, a surly man named Abe, glared at Frank as he entered and pointedly dropped his hand below the scarred counter. The message was clear: "I am now holding a weapon. I'll let you guess what kind. But if you hassle me, I'll give you a very close look."

"I don't want any trouble, Abe," said Frank, holding up his hands. "I'm just here to see Billy."

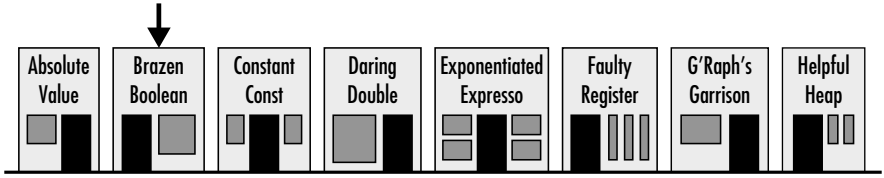
"Well, Billy ain't here," said the barman.

Frank almost smiled in relief. "Then I'll be on my way," he said.

Abe gave a curt nod and watched Frank leave, his hand still under the counter.

Frank took a couple of deep breaths and shook his head in the cool air. Abe held a grudge longer than anyone else Frank had ever met. Then again, Frank had arrested four of his siblings.

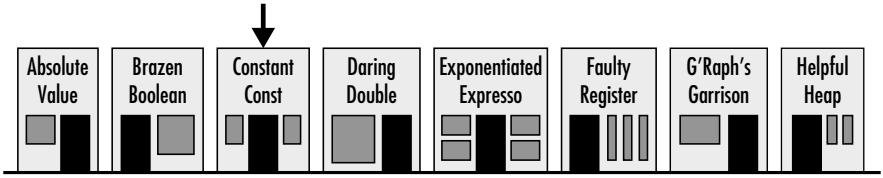
The next establishment on the street was The Brazen Boolean, a modern coffee shop decorated in typical Boolean style—stark black and white. The inhabitants of the City of Bool were renowned for their fanatic devotion to the absolute concepts of logic, viewing everything as either True or False. They made good witnesses. As the only Boolean café in town, The Brazen Boolean was a haven for expats. After all, either you were a Boolean or you weren't.



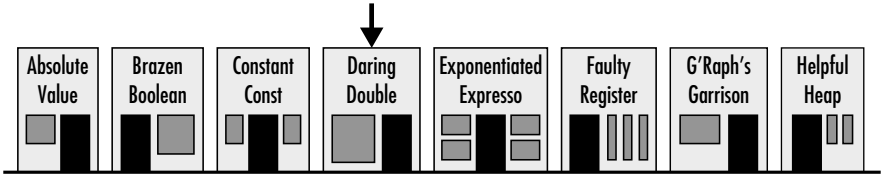
Frank popped his head in the door and asked everyone in general, "Is Billy here?" There was a brief silence as twenty pairs of eyes carefully scanned every inch of the cafe. Booleans wouldn't answer a question until they were absolutely sure.

"No," came the precise reply.

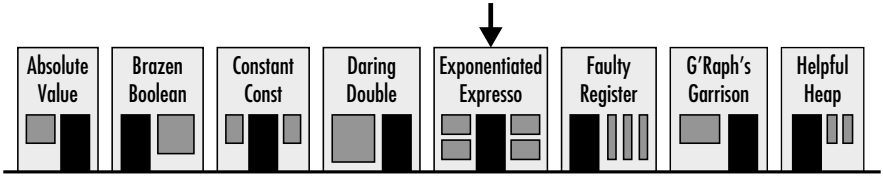
Frank continued his exhaustive search.



The third and fourth shops proved equally fruitless, although significantly more pleasant. The bartender of the Constant Const greeted Frank warmly and invited him in to reminisce about the good old days together, which was odd considering Frank had only met him the month before. And the crowd of the Daring Double, a notoriously loud wizards' hangout, cheered at each new arrival and sang happily over their steaming mugs.



Frank found Billy in the fifth shop, the Exponentiated Expresso. It was by far the loudest and tackiest coffee shop on the street, but it managed to draw the most devoted following on account of its triply caffeinated beans. On a good day, every table would be packed with jittery people who seemed to think the key to a good conversation was volume.



This morning, the Exponentiated Expresso hosted a comparatively subdued crowd. Only a handful of tables were occupied, and most of those by lone coffee drinkers who shook and mumbled quietly to themselves.

Billy sat at a central table, leaning awkwardly toward a nearby conversation. Nobody seemed to notice him. Frank had even missed him on his first scan of the room.

"Billy!" Frank called.

Billy jumped up guiltily. "Frank?" He grinned, happy that someone had acknowledged him, and sat back down. "Pull up a chair."

"I'm looking for some information," explained Frank as he took a seat across from Billy.

"Could be that I have some," said Billy. "I have such a hard time remembering these days," he said, glancing toward a long-empty mug that probably wasn't his.

Frank signaled the barista, who soon placed a fresh mug on the table. "Remember anything about a theft at the police station?" Frank asked Billy.

Billy's eyes widened and he flinched. "A robbery, you say?" he asked unconvincingly. His eyes darted around the room, but, as always, nobody paid him any attention.

Frank laid two gold pieces on the table, ignoring the sour feeling in his gut. He couldn't afford to spend this type of money, especially without knowing if he was paying for a lead or idle gossip. But he'd known this wasn't going to be cheap. He leaned in close. "Two nights ago," he said quietly, "the thieves took a whole pile of documents."

"Doesn't sound like the sort of thing that would be healthy to remember," said Billy. He eyed the gold pieces. "Afraid you're asking the wrong guy, Frank."

"That's gold," Frank growled.

"Sorry. I can't help you," Billy said. He surveyed the room again before adding, "Even if I did know something about a robbery, it's

the sort of thing I would try to forget. Even if I did know something small, like who might have helped with logistics, it's not worth the risk of waking up to find my shoes packed with yak dung."

Frank stared, but Billy had gone silent. For someone who made a living sharing information, Billy had an odd habit of not saying things."Yak dung?" asked Frank.

Billy nodded, but didn't offer any more.

"You couldn't be more specific, could you?" asked Frank. "Are we talking about Northern or Southern yaks?"

"Does it matter?" asked Billy. "The point is that if I knew anything about who arranged transportation, I wouldn't remember it. Especially not if those people happened to have a large farm about five miles out of town where they could easily make someone disappear. And very doubly especially if the family that owns the farm has a history of illegal activity and an unhealthy sense of humor. Nope. It definitely wouldn't be healthy to remember anything in that case."

"Too bad," Frank said with a smile. "Maybe next time then." He nodded toward the coins. "Incentive to remember things in the future."

With that, Frank stood and strode from the Exponentiated Expresso. He turned left and continued up the street. Once off Three Bit Lane, he could swing around and make for Crannock's farm—the only farm remotely matching Billy's description.

As he passed the Faulty Register, he noticed a shadow dart into a nearby alley. He cursed under his breath, but kept going. Of course he had a tail already; the captain hadn't exactly been discreet about his visit.

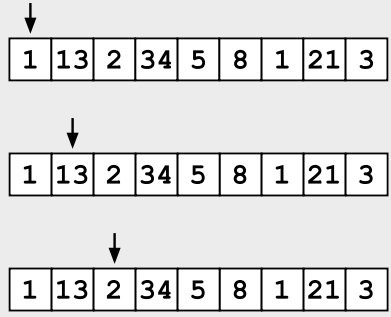
But by the time he left the city and was on the rough dirt lane to Crannock's farm, he found himself in a good mood. Billy hadn't given him much, but even a little information could mean the difference between an efficient search algorithm and an exhaustive one.

POLICE ALGORITHMS 101: EXHAUSTIVE SEARCH
Excerpt from Professor Drecker's Lecture

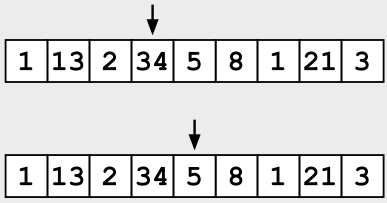
An exhaustive search algorithm searches every possibility in the entire search space for the target value. The most common exhaustive search is a *linear search*, which simply checks all the different possibilities in order.

Consider what happens when you chase a robber into the second-floor hallway of an abandoned hotel. The hall has 30 doors, all of them closed. If you've followed correct police procedures, your partner has already blocked off the opposite staircase, and the robber is trapped somewhere on that floor. How do you find him? Do you pick random doors, running back and forth until you get lucky? No! You search down the hall, kicking in one door at a time.

Or consider an algorithm that scans a list of numbers (an *array*), searching for a target value. The algorithm moves along the list from number to number, checking each value in turn so as not to miss any, and stops when it reaches the target. If we are searching an array for the number 5, then the search would progress as follows:



continued



The advantage of linear search algorithms is that they are simple to implement in the field and they work even on unstructured data. You don't have to make any assumptions about which room the robber chose; you just check everything. The downside is that exhaustive algorithms are often not the most efficient algorithm if the data has structure that can be used. If you know where the robber went, you can save yourself from kicking a lot of doors by using that information.

The key to efficient algorithms is information!

— 3 —
Arrays and Indexes on a Criminal's Farm

Frank swore aloud when he saw the police horse tied outside Crannock's house. Since the captain had gone so far as to hire Frank in person, he hadn't expected to run into any officers. If the captain didn't trust his officers, either they were under suspicion—so he'd shuffle them to some case far across the city—or they simply weren't good enough. But, from the looks of it, someone was on the case and Frank was already behind.

He slid through the open front door and joined the officer and Mr. Crannock in the foyer. Mr. Crannock shot him a disgusted look but didn't seem surprised to see him. The officer, however, seemed caught off guard.

"Who are you?" she demanded, turning on him with parchment and quill in hand.

Frank ignored her. "Mr. Crannock," he said. "So wonderful to see you again."

"Come to harass us, too?" Crannock asked. "You're not welcome here, Frank."

"I'm not looking for a welcome," replied Frank. "I'm looking for your wife. I have a few simple questions for her."

The officer stared at him. "Frank?" she asked. "Frank Runtime? Former detective turned private eye? What are you doing here? Someone lose a pet dragon?" she scoffed.

Frank ignored her again. "Your wife, Mr. Crannock. Where can I find her?"

The old man threw up his hands. "She didn't do anything! She's gone straight, you know. For real this time." His acting wasn't half-bad for an amateur.

Frank smiled; he knew its effect was unnerving. Sure enough, Crannock cringed.

"I know that, Mr. Crannock. I'm here to tap her professional knowledge. Or I could just leave the conversation to . . ."

"Officer Notation," the young officer snapped. "And this is *my* investigation."

That was a lie. Officers always worked these investigations in pairs. More importantly, Frank recognized Notation's name from the duty roster the captain had given him. She had been at the station on the night of the crime.

"Officer Notation," Frank said. "Who said I'm here on an investigation? Maybe I'm simply searching for a lost dragon."

She scowled.

There was a commotion coming from the back of the house. Someone called for Crannock, but was cut off by a loud braying noise. "My wife is with the horses," Mr. Crannock said impatiently. "Barn #2. Now go on, get out of my house!" Crannock waved them toward the front door and scurried away through the back.

"Thank you," Frank called as he turned to leave. "Always a pleasure, Mr. Crannock."

Officer Notation followed Frank across the yard. She walked hard, stomping her anger into the ground. "Do you know where you're going?" she asked.

"Barn #2," answered Frank.

"I know that," seethed Notation. "But where is barn #2?"

Frank stopped and turned to her. "Just out of the academy, Notation?" he asked.

"What?"

"Only a rookie would ask a search question like that. Didn't you take Police Procedures and Data Structures? Or have they replaced that course with something less rigorous—Introduction to Turtle Graphics, perhaps?"

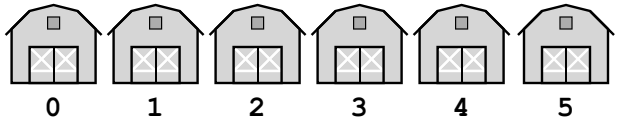
Notation seemed taken aback. "Of course I took Police Procedures and Data Structures," she said, though she sounded uncertain. "But what I meant was—"

Frank cut her off, "Then you know about arrays and indexes."

"Yes, but—" started Notation.

"Finding a barn on a farm is a simple enough search task," Frank interrupted again. "We could use an exhaustive search to check each building. *FOR EACH building on the farm: check if it is barn #2.* Back in my day, you learned that search on the first day of Police Algorithms.

"But we can do better here. The Crannocks have six barns in a nice line—just like a giant array. Mr. Crannock was kind enough to supply us with the barn number, the index into that array. All we have to do is walk to the corresponding barn."



"That's not what I meant!" shouted Notation, waving her arms. "I know how to use the index of an array. I know that we only have to walk up to the barn with a giant #2 outside. I graduated first in my class in both Data Structures *and* Police Algorithms, so don't lecture me on the correct use of arrays."

"Well, you asked," Frank replied.

"What I was asking is: Do you know where this wonderful array of barns is located?"

"Of course you were," said Frank. He began walking again. "You still sound like a rookie, though, quoting class rank."

"Where are the barns?" shouted the officer, stamping to catch up.

Frank shot her a smile over his shoulder. "Over this hill."

As Frank had learned years ago, the Crannock family embraced the concept of arrays with an almost fanatical devotion. They organized everything into linear structures with clearly labeled indexes for each element. As he passed barn #0, Frank noted 15 pig troughs, each capable of storing one serving of food. A farm hand was iterating down the line and ladling out the next meal into each array location.

Frank and Officer Notation moved on to barn #2, labeled with a sign outside its door. Mrs. Crannock's icy greeting was almost pleasant, compared to previous encounters; she hadn't even thrown anything . . . yet.

"What do you want?" Mrs. Crannock demanded.

"Mrs. Crannock," Notation cut in before Frank could steal her witness. "I was hoping I could ask you a few questions."

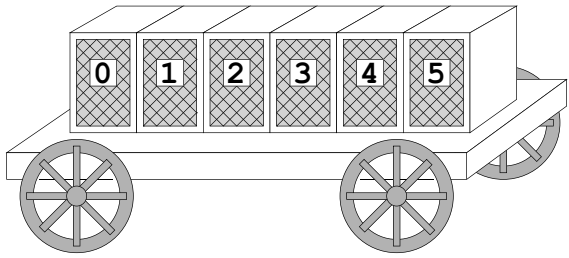
Frank let Officer Notation ask the questions. Billy's clue hadn't yielded anything more than a lead to the farm, but Notation appeared to be working from a better collection of clues.

Mrs. Crannock sneered and spat on the ground. "I didn't do anything," she said. "I've gone straight, you know."

"I'm not here to arrest you," said Notation. "I need to ask you about a certain donkey cart—the ArrayCart?"

A flicker of doubt went through Frank. Could Officer Notation be here for a different case? He doubted it. His gut told him that she was after the lost documents, and he had learned to trust his gut.

"The ArrayCart," said Mrs. Crannock suspiciously, though with the barest note of pride. "My own invention. Based it off of an array. It's got individual storage pens for our animals. Each pen stores exactly one animal. Since they all have separate doors, you can walk up to any pen and take an animal out or put one in. Easy access to any storage location. Saves hours of wrangling."



"It's quite ingenious," Officer Notation conceded. "You've found a way to apply the concept of arrays and indexes to livestock transportation."

"And that's just the beginning," added Mrs. Crannock. "I'm working with a certain *wizard* on a completely new type of ArrayCart—one with magical pointers! I bet they'd be perfect for the police force. Tell your captain that I can give him a good price."

Frank had to hand it to Notation. The surest way to get a Crannock talking was to bring up arrays.

"You have a few ArrayCarts that you rent out now. Is that correct?" probed Officer Notation.

Mrs. Crannock's eyes became instantly cold. "It's a legitimate business. We pay our taxes."

Frank held back a derisive snort.

"Did you happen to rent an ArrayCart to anyone two nights ago?" pressed Officer Notation. "A smaller model with six pens."

"I might've," said Mrs. Crannock. Her cold demeanor was creeping toward hostile.

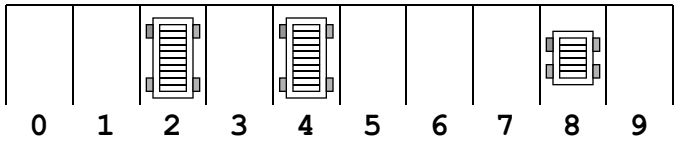
"Do you have a record of who rented it?" asked the officer.

"No," said Mrs. Crannock. "We shred the records once the carts are returned. I don't happen to recall who rented that one."

It seemed like Billy's hint had paid off. If you were a criminal in need of transportation, there were few places that would rent you a cart, and fewer still that would forget your name afterward. Mrs. Crannock may have claimed to have gone straight, but apparently she still, at the very least, provided a valuable service to her former associates.

"Are you sure you don't remember anything about your customer?" prompted Officer Notation, but Frank knew it was pointless. He had once questioned her for three hours about a stolen yak. She hadn't given him a single peep, despite being the one who had been robbed. Mrs. Crannock wouldn't talk.

While Officer Notation tried a few variations of the same question, Frank quietly slipped out of the barn and found the cart lot. As he expected, the lot was organized as an array with 10 labeled parking spots. Only spots #2, #4, and #8 were occupied. The carts in positions #2 and #4 had 10 pens apiece, so were too large to fit Notation's description. But slot #8 held a six-pen ArrayCart, its wheels still coated with fresh mud.



After a quick glance around, Frank heaved himself into the back of the six-pen ArrayCart. A scattering of straw covered the floor, but the cart was otherwise empty. Frank opened each pen in turn, scanning the empty storage spaces for any clues. Then, getting down on his hands and knees, he sifted through the straw until he found a few scraps of parchment.

He collected six tiny pieces in all, probably corners that had caught on nails as the scrolls were unloaded. Only two of the pieces contained writing, and those appeared to be from ledgers. It wasn't exactly a solid lead, but it tied the cart to the crime.

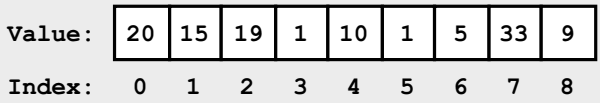
Frank moved his search to the front of the cart, carefully inspecting everything around the driver's seat. The seat itself gave him his first real clue. There he found a few black and orange threads caught by the seat's splintered wood. From the vividness of the colors alone, Frank could tell the cloak must have been new. Satisfied, he pocketed the threads and stepped down from the cart.

Only when a gust of fresh air hit him did he notice he'd been holding his breath. Around the cart was a stench of rotting fish. He sniffed lightly, following the smell, and arrived at the mud-caked wheels. He took a great, deep sniff and immediately regretted it. The smell of rotting eel emanating from the mud was as unmistakable as it was unpleasant.

Frank half-smiled, half-gagged as he staggered back from the cart. He might not know who had rented it, but now he knew where it had been.

POLICE ALGORITHMS 101: ARRAYS
Excerpt from Professor Drecker's Lecture

Arrays are simple data structures that allow you to store multiple values. An array is like a row of bins. Each bin can store a single piece of information, such as a number or a character.



The structure of an array means you can access any value (or element) within the array, whether to write to it or read from it, by specifying its location, or *index*, within the array. Many programming languages use 0-indexed arrays, which means the first value of the array resides at index 0, the second at index 1, and so forth. Commonly, you reference the value at index i of array A as $A[i]$; for example, the third element of array A would be $A[2]$ and equal to 19.

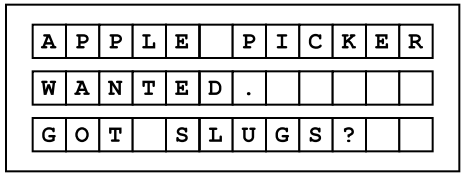
You likely recognize this structure from your introductory tour of the holding cells in the capital's police station yesterday. The king personally suggested the use of indexed, single-person cells to streamline the retrieval of prisoners. Each station is equipped with an array of four to eight holding cells, depending on the size of the local criminal population.

— 4 —
Strings and Hidden Messages

Frank shook off Officer Notation and left through the back gate of the farm, where a large sign faced the road. For years the Crannocks had used this sign to broadcast coded messages about various illegal activities. These days it was something of a tourist attraction for visiting criminals—a place thugs took their younger protégés and gathered to reminisce about stories that invariably began "Back in my day . . . "

The sign itself was an AnyText model. It held 3 arrays of letters, each array with 12 slots. Each letter, space, or punctuation mark took up a single slot in an array, meaning the board could hold a total of 36 individual characters—enough to advertise a whole range of illegal activities. Every Monday morning, one of the Crannocks would drag a basket of letters to the sign and individually place the appropriate character in each slot of the array.

During his first week on the force, Frank's partner had brought him out here to "check the board." The message at the time—*Apple picker wanted. Got slugs?*—sounded innocuous enough to Frank. The Crannocks were looking for an apple picker to help with the harvest and were offering to get rid of people's slugs. When he said this to his partner, a 20-year veteran, she laughed.



"That's what they want you to think," Detective Rossile explained. "You have to look beyond the obvious meaning and see what the criminal mind would see. In this case, *Apple picker wanted* indicates that they are trying to hire a petty thief. Someone who would steal apples from a cart or such."

"And the slugs?" Frank asked.

"Illegal slug racing," she replied. "They hold races here every few months. You'll get to know that one."

Thus, Frank had learned to check the Crannocks' board weekly to get a pulse on the criminal world. After the first few months, he had learned to decipher most of the codes. *Farmhands* meant henchmen, with additional modifiers if strength, brutality, or just plain numbers were needed. A *print artist* referred to a forger, while a *vocal artist* was a con man, and so forth. The phrase *a flock of chickens* had stumped Frank for a few days before Rossile translated it as "a large number of warm bodies to run around noisily and cause a distraction; no intelligence required."

By the end of his first year, Frank had become an expert at reading the board. The only time in the past few years that Frank had a hard time deciphering a criminal tip from the board was during the wizard Exponentious's attack on the kingdom. Exponentious had unleashed the Spell of Incorrect Indexes on all the kingdom's ArrayDesignBoards. As its name implied, the spell changed the indexes, so the locations Mrs. Crannock thought she was setting the letters to were wrong. For a week, the Crannocks' board held gibberish.

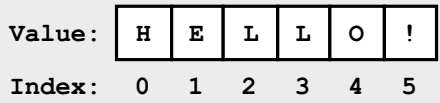
D	E	F		V	E	I	N	S			E
D	I	Z					W	A	R		
D	E	N	T						A	W	

Since the spell only mixed up letters within an array and the AnyText model was implemented as three separate arrays, Frank had to unscramble each line individually. He puzzled out the message: *Defensive wizard wanted.*

Today, though, the message was clear. In fact, it was the least subtle message he had ever seen on the Crannocks' board. It read *ArrayCarts for rent. No questions.*

POLICE ALGORITHMS 101: STRINGS
Excerpts from Professor Drecker's Lecture

Arrays don't just store lists of numbers; they can also be used to store strings of text characters. Many programming languages implement strings using arrays. Each block in the array holds a single character, which can be a letter, number, symbol, or space. As with arrays of other data, characters in these strings can be accessed directly through their index in the array.



During your career in the police force, you will come to know this representation of text *very* well. All standard police forms require officers to record their names within a 32-block array at the top of *each* page. In a typical month, you will fill in over 400 such arrays.

— 5 —
Binary Search for a Smuggler's Ship

The port of Usb was little more than a fishing village. A dozen weathered buildings clustered around the end of a single long pier. A few pockets of meager activity surrounded the most recent arrivals, but otherwise the town was reassuringly quiet.

Frank headed straight for the Crab's Pinch, a fisherman's bar renowned for its clam chowder and Wednesday night sea shanty contests. With any luck, one of his contacts would turn up before the day was out. After all, the Crab's Pinch was the only place to go in Usb. So Frank planted himself at a table in the back corner, ordered the chowder, and waited.

It wasn't long before a freelance smuggler named Mavis entered the dank little bar. Careful by nature, Mavis had never technically been convicted of a crime, though it was well-known that she'd once set her own ship on fire to destroy evidence. Frank got along with her well enough, at least once he'd left the force, and they even exchanged the occasional scrap of information.

Frank, having nursed his chowder for a solid hour, finally pushed away his bowl and motioned to Mavis. She hesitated a moment by the door before jostling her way through the bar.

"Mavis," said Frank as she joined him in the corner, "how are you?"

"I was doing a lot better 10 minutes ago," she spat.

Before Frank could ask, Officer Notation strode through the door and held up her hands. "Ladies and gentlemen," she called. "If I could have your attention for a moment. I'm looking for a cart that came through here two nights ago."

Frank cursed under his breath. So much for his lead.

"I come in from the dawn run, hoping for a bowl of hot chowder and a few minutes of peace," Mavis complained. "Instead I get this copper clammering about donkey carts."

Frank laughed dryly. "And until she goes away, you can't unload your cargo. Right?"

Mavis scowled at him but didn't object. Usb had never found success in either the fishing or shipping industries. The port did,

however, appeal to those criminals concerned with moving merchandise without dealing with nosy government officials. Frank would wager a month's rent that there wasn't a single ship at dock that wasn't smuggling something.

"Do you know anything about the cart?" Frank dropped his voice to just above a whisper.

Mavis shrugged. "There's always carts on the docks. This is a port, Frank. People move things."

"This is a special cart," Frank pressed. "A bunch of individual animal pens, like a giant array on wheels."

"Sounds fancy," said Mavis. "But I haven't heard of any ships moving animals. I might have heard a rumor about a crate or two of miniature turtles, but nothing large enough to need a pen. You sure it came through here?"

Frank nodded. The smell had been like a fish-scented air freshener in an outhouse, and few places smelled as bad as Usb.

"Anybody casting off at that time?" he asked. If the thieves had transported the stolen documents this far, they wouldn't have waited around.

"Only the *Retry Loop*," said Mavis. "And I'm only telling you that because it's public knowledge. I don't know what it was carrying, and I don't care."

"Do you know when it returns?" asked Frank.

"Got back into port 19 hours ago," replied Mavis. "Don't know what it was carrying then either."

Frank smiled widely. "Sounds like it's time for me to take a stroll around town," he said.

Mavis smiled halfheartedly at him and turned to flag down a waiter.

Frank made it less than 20 meters down the pier before Officer Notation marched up beside him.

"Mr. Runtime, this is my investigation," she began. "If you have information—"

Frank stopped, causing her to pull up short. "What exactly are you investigating, Officer?" he asked.

It was better than Frank had hoped. Notation opened and closed her mouth a few times as a red flush spread up her neck.

"The captain doesn't know you're here, does he?" Frank asked. "This isn't exactly an *official* investigation."

"I don't know what you're—" started Officer Notation, but Frank cut her off.

"Cut the act," he said. "The fact you're out here alone is all the proof I need. You're running this investigation on your own time. The question is, why?"

The flush had now finished its ascent of Officer Notation's face. Her ears burned a particularly vivid shade of red.

"That's none of your concern," she said.

"It is when the captain comes to me because he can't trust his own officers," Frank replied calmly.

"The *captain* hired a washed-up gumshoe like *you*?"

"Yes. Because he can trust *me*."

Officer Notation's face grew hard and her eyes burned. For a second, Frank thought she might end this conversation with her billy club. But almost as quickly as her anger had flared, it deflated.

"I need to recover those documents," she said mournfully. "It was my fault—I was on guard duty that night."

"I see," said Frank thoughtfully.

"I need to recover those documents," repeated Officer Notation, sounding agitated. "I've only been on the force for a few months and—"

Frank cut her off and gave her what he hoped was a reassuring smile. This was what he had expected. Rookies rarely dealt well with their first mistakes, and Notation seemed more tightly wound than most. "We're looking for the *Retry Loop*," he said. "The Crannocks' cart unloaded something there the night of the robbery. The ship docked 19 hours ago."

He didn't trust her, of course, but he wanted to keep her close, keep an eye on her. The fact that she had found the Crannocks meant she knew more than she had put in her report. Something was missing from her story, and he needed to find out what else she knew.

"We better get started," said Notation, looking worriedly down the pier. "There are a lot of ships to check. Should we start at the front?" As most of the vessels in port belonged to smugglers, none of them displayed identification. They would have to ask each ship's name in turn.

"We can do better than that," Frank explained. "The harbormaster is fanatical about organization. He insists that the docked ships be sorted in order of their arrival time. The newest arrival gets a prime spot near town, where the crew can easily load and unload, but when a new ship arrives, the rest of them are forced to shift down to give it space in front."

"That's absurd," protested Notation. "What a tremendous amount of wasted effort. Why would he do that?"

Frank chuckled. "He claims it's for efficiency, but anyone who's spent a week in Usb knows the truth. The harbormaster can't stand the smell of rotting fish. Ships that remain in harbor without selling their loads become, well . . . fragrant. The harbormaster's organizational scheme moves the ones that have been here longer away from his shack."

Officer Notation stared at him. "Are you serious?" she asked finally.

Frank chuckled again. "Yes. You'll start picking up these useful bits of information, too, once you've walked the beat awhile. The point is that we know the ships are in sorted order and we know the *Retry Loop* has been here for 19 hours, so we can just do a *binary search*.

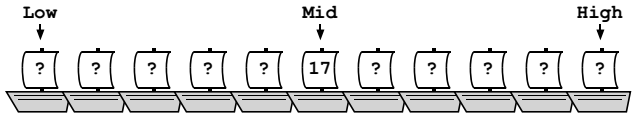
"Our target value is 19, and our algorithm is binary search. Right now the search space is that whole line of ships, so we already have an upper and lower bound. If we use inclusive bounds, our lower

bound is the first ship and our upper bound is the last ship. If the *Retry Loop* is here, it obviously can't be in front of the first ship or after the last ship.

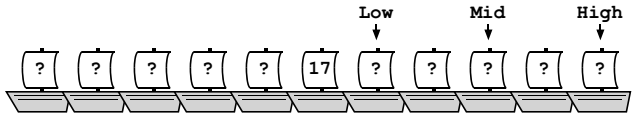
"So we start with the middle ship and ask how long it's been in port. If it's been there less than 19 hours, then it must come before the *Retry Loop*. That will split our search space in two. And—"

"If it's been there more than 19 hours, then it must come after the *Retry Loop*," interrupted Notation. "I know about binary search. My Police Algorithms final was just two and a half months ago."

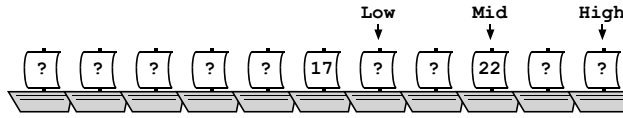
With that, the two of them set off in search of the *Retry Loop*. The middle ship, a yellow schooner that smelled oddly of bananas, had been in port for 17 hours.



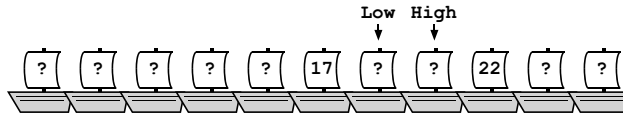
That meant they could rule out the half of the ships at the front, including the middle ship. Frank adjusted the lower bound to the first ship that *could* be the *Retry Loop,* one ship past the yellow schooner.



With the reduced search space, they chose a new middle point. It took a while to convince the captain of the next ship that they weren't undercover customs officials. After 10 minutes, Notation shoved her badge under the captain's nose, and his tone changed immediately to an irate whine as he informed them that his ship, the *Corrupt Packet*, had been stuck in port for 22 agonizing hours. He demanded they speak to the harbormaster on his behalf.

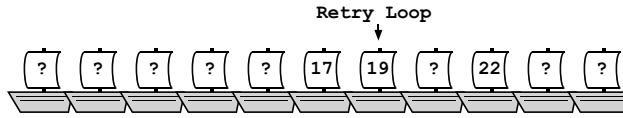


Since their target was 19 hours, they knew the *Retry Loop* would have to come before the *Corrupt Packet*. They changed the bounds again so that the ship to the left of the *Corrupt Packet* was now the upper bound.



This left only two ships in the search range; they were rapidly nearing the end of the search. If neither of these ships was the *Retry Loop*, they would know for certain that it had left port, as once there were no more elements in the search space, they could rule out the entire search space.

Since there were only two ships left, they could choose either as their new middle point. Going with his gut, Frank picked the earlier ship, which happened to also be their lower bound. A quick chat with a crewmember loitering on the pier confirmed that the ship was indeed the *Retry Loop* and it had been in port for 19 hours.



"Now what?" asked Officer Notation as they stood watching the ship.

"We use your shiny badge again," Frank replied.

POLICE ALGORITHMS 101: BINARY SEARCH
Excerpts from Professor Drecker's Lecture

A binary search algorithm is used to efficiently find a target value v in a sorted array A. Unlike in a linear scan, a binary search uses information about the structure of the data to make the search more efficient. The key to efficient algorithms is information. In this case, we use the fact that the array is sorted in increasing order:

$A[i] \leq A[j]$ for any pair of indexes i and j such that $i < j$

This might not seem like a lot of information, but it's enough to make the search more efficient.

The binary search algorithm works by repeatedly dividing the search space in half and limiting the search to only one of those halves. The algorithm limits the active search space by tracking two bounds. The upper bound (*IndexHigh*) marks the highest index of the array that's part of the active search space. The lower bound (*IndexLow*) marks the lowest index. Throughout the algorithm, if the target value is in the array, we guarantee the following:

$$A[IndexLow] \leq v \leq A[IndexHigh]$$

At each step in the search, we check the value halfway between the lower and upper bounds:

$$IndexMid = \frac{IndexHigh + IndexLow}{2}$$

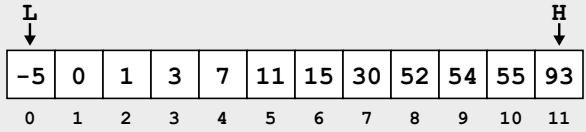
We can then compare the value at this middle location, $A[IndexMid]$, with the target value, v. If the middle point is less than the target value, $A[IndexMid] < v$, we know that the target value must lie after the middle index. This allows

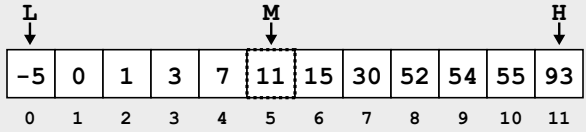
us to chop the search space in half again by making *IndexLow* = *IndexMid* + 1.

If the middle point is greater than the target value, $A[IndexMid] > v$, we know the target value must lie before the middle index, which allows us to chop the search space in half by making *IndexHigh* = *IndexMid* – 1.

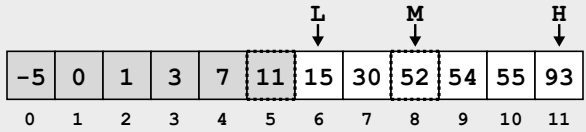
Of course, if we find $A[IndexMid]$ equals v, we can immediately conclude the search. We found the target.

Let's consider searching the following (sorted) array for the value 15. The boxes with dotted outlines correspond to the values the algorithm has checked, and the shaded elements are ones that have been eliminated from the search.



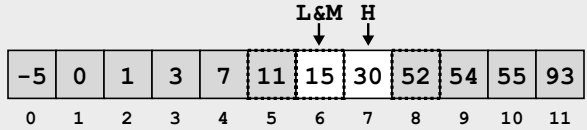


The first midpoint check finds a value of 11, which is less than our target value of 15. Since we know the array is sorted in increasing order, we can rule out the midpoint and anything before it. We move our lower bound index appropriately (*IndexLow* = *IndexMid* + 1).



continued

Similarly, after the second comparison, we find a midpoint value of 52, which is greater than the target value. We can rule out the midpoint and everything after it. We move our upper bound index (*IndexHigh* = *IndexMid* – 1).



Note that even though the lower bound's index pointed to the target value ($v = 15$) for several iterations, we continued the search until the midpoint pointed to the target value. This is because our search checks only the value at the midpoint. We don't check the values at the lower or upper indexes until the midpoint reaches them.

What happens if the target value is not in the array? As the search progresses, the bounds will move closer until there are no unexplored values between them. Since we are always moving one of the bounds *past* the midpoint index, we can stop the search when *IndexHigh* < *IndexLow*. At that point we can guarantee the target value is not in the array.

— 6 —
Binary Search for Clues

"Food inspectors," Frank called out as he and Officer Notation strode up the narrow gangplank and onto the ship. At Frank's instigation, Notation waved her badge in a blur, too fast for anyone to read.

"Food inspectors?" asked a crew member. "We aren't transporting any food."

Frank looked the man over. He wasn't an officer or hired security, probably just a sailor who had taken charge while the officers were away. It wasn't uncommon. Smugglers rarely employed guards to watch their ships. It drew too much attention.

Frank turned on the sailor, growling out his words. "We'll see about that. I'm told there's a load of rotting eels on this dock, and I intend to find them."

"Eels?" The sailor was clearly out of his depth.

"*Rotting* eels," Frank shot back. "We're going down to check the stores." Then, without waiting for a response, he strode to the hatch leading below deck.

Notation hurried after him.

"We don't have much time before they get the captain. We need to find the logbook," Frank said as he climbed down the ladder. The

logbook would contain a manifest of items shipped and a list of ports visited. The manifest would be fake, of course. Smuggling ships never documented their true cargo. But with any luck, he could read between the lies and find a clue.

Officer Notation found the logbook at the back of the hold and pulled it out. Frank checked the cover and swore:

> Manifest and Log of the *Retry Loop*
> Captain: A. James
> Home Port: Usb
> Owner: Vinettees Shipping Group LLC

After months of successfully avoiding the Vinettees, Frank had walked right onto one of their ships. He found himself reflexively scanning the hold for hidden henchmen, stashes of weaponized farm equipment, or evidence of a slug racetrack. Frank discarded the last possibility—everyone knew slugs wouldn't race on a ship; it had something to do with being surrounded by large quantities of saltwater.

He shook his head and focused on the problem at hand. Frank had to find a clue before the Vinettees knew he was on the ship, or he might not get back off again. He turned to the end of the book and started flipping toward the front, one page at a time.

"What are you doing?" asked Notation.

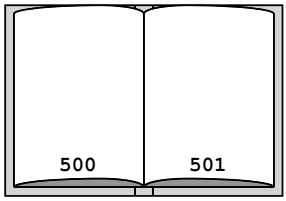
"Looking for the last entry," said Frank.

"One page at a time?" asked Notation. "There's got to be a thousand pages. Why don't you use binary search again? We just used it two minutes ago."

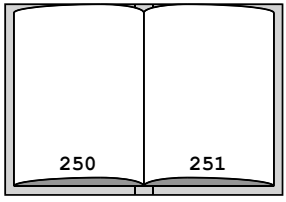
Frank paused. He wasn't looking for a specific page number, but he could still use a binary search to find the last entry. He would just refine the search bounds depending on whether the current page had text or not.

"Okay. Binary search," he agreed.

He opened to the last page again and confirmed the book had exactly 1,000 pages, giving him a lower bound of page 1 and an upper bound of page 1,000. He added the numbers, divided by 2, and computed a midpoint of 500. He flipped to that page.

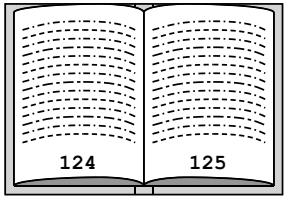


Pages 500 and 501 were both blank, so Frank knew the last written page was on or before 499—his new upper bound. After another midpoint computation, he flipped to page 250. Again it was blank.

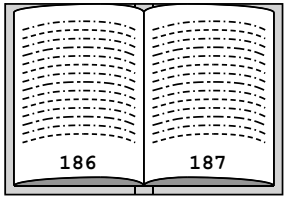


"Looks like a new book," added Notation. "Good thing you didn't keep going from the back."

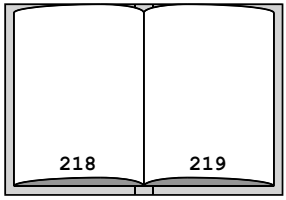
Frank didn't bother replying. With a lower bound of 1 and an upper bound of 249, he computed a midpoint of 125. This time he found writing, so he adjusted his lower bound accordingly to 125.



"187," supplied Notation before Frank could finish the midpoint computation in his head. He turned to 187, again finding writing and adjusting his lower bound.



"218," said Notation. The pages were blank, so Frank adjusted his lower and upper bounds to 187 and 217, respectively.



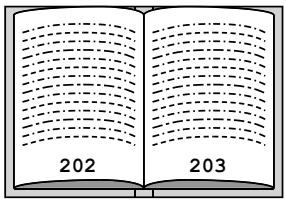
"202," said Notation before Frank had even finished adding the upper and lower bounds.

"How are you doing that so fast?" asked Frank.

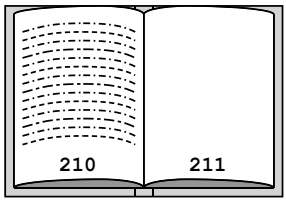
"Practice," she replied. "We used to have binary search competitions at the academy whenever we needed a break from studying. I was undefeated."

Frank shook his head. "Sounds like a wild time," he muttered.

Pages 202 and 203 were filled. “210,” supplied Notation.



At page 210, they finally found the last entry, detailing the *Retry Loop*’s last voyage. “What now?” asked Notation.



“We search for an interesting package or port. On the last voyage, they made about 70 entries. We’ll have to scan through them.”

“Exhaustive search?” asked Notation. “Can’t we use something more efficient? Aren’t the entries sorted by pickup and delivery time?”

“The sorting doesn’t help us here,” answered Frank. “We don’t know the time. Sorted data only helps when it’s sorted by a useful dimension. They didn’t bother to sort by suspiciousness. Go figure.”

“Oh. It’s the Weather Records problem,” said Notation.

“What problem?” asked Frank.

“It’s an illustration of how sorting data by the wrong value doesn’t help a search,” explained Notation. “Professor Drecker gave the example of finding the coldest day in the last 10 years. If the logs are sorted by day, you can use binary search to efficiently find any specific day. But that doesn’t help us to find the coldest day, so we’d still have to scan through all the data. I hadn’t expected to see such a clear example outside of class, though.”

"Welcome to the real world," said Frank. "Out here, you have to check when the structure in the data is helpful to you and when it isn't. Don't worry, it's a common rookie mistake."

He could see Notation bristle at his words and tried not to enjoy her reaction too much. Every rookie came out of the academy thinking they knew it all, and every one had a lot to learn. Notation was getting off easy with a lecture. His own education with binary search had involved hours of scooping through barrels of pig waste while he questioned his career choice.

After about three minutes, they found their only clue. The *Retry Loop* had recently made two suspicious stops, at the Port of Mudwall and Frayed Cable Island. Even for smugglers, these were strange destinations. The Port of Mudwall boasted little trade beyond its outlying mud farms. And Frayed Cable Island was even more desolate; the small, rocky island possessed only a single building—the now abandoned Iron Ring Prison.

"There," said Frank, pointing. "That's where they took your documents. Either the Port of Mudwall or Frayed Cable Island. They probably dropped the documents off at one and picked up the payment at the other."

"How do you know?" asked Notation. She looked skeptical. "Shouldn't we consider all the ports as—"

Frank cut her off. "No time to check them all." He didn't elaborate. He was using his own brand of algorithm now, the heuristic searches that had gotten him in trouble with the captain in the past. But he had a gut feeling, and Frank had learned to trust his gut.

"Are you sure that—" Notation began, but was cut off by noises above them.

Frank couldn't make out the words, but he could recognize the tone clearly. Trouble was on its way.

POLICE ALGORITHMS 101: BINARY SEARCH II
Excerpts from Professor Drecker's Lecture

The key to efficient algorithms is information. In the case of binary search, we require that the data be sorted and that we have information on how that data is sorted. In order to rule out (or prune) large regions of the search space, the algorithm must be able to guarantee that the target value can't be in that region. We can only do that if we know how the values behave as we move along the array. In computational problems, we say the array is sorted if all its values are arranged in increasing (or decreasing) order.

However, just because the data is sorted by *one* dimension doesn't mean that you will be able to binary search along another dimension. Say you're searching an accounting ledger for clues. Ledgers are sorted by transaction number, which indicates when the transaction was *recorded*. This means that the transaction number for each entry will be less than the transaction number for the following entry. If the current entry has a transaction number of 105, we know that all entries before it will have transaction numbers less than 105 and all entries after it will have transaction numbers greater than 105.

101	August 16	Zed's Coffee	8.00
102	August 15	Bob's Pizza	20.00
103	August 15	Wands and More	150.00
104	August 15	Spell Shoppe	100.00
105	August 16	Zed's Coffee	8.00
106	August 16	Spell Shoppe	50.00
107	August 17	Zed's Coffee	8.00
108	August 17	Hospital	250.00

continued

However, this also means that the entries in other fields, such as the actual date of the transaction, the mechant's name, or the transaction amount, are *not* in sorted order. What if you're interested in finding transactions above a certain suspicious amount or with a known weapons merchant? Does the sorting help you here? No, you would still find yourself using an exhaustive linear search. Knowing that transaction 105 was at Zed's Coffee doesn't tell you anything about the merchants or amounts in transactions before or after that one.

Similarly, if you sorted the ledger in increasing order of transaction amount, this would allow you to quickly find all transactions costing $250, but it would not help you search for a given transaction date, ID, or merchant.

— 7 —
Adapting Algorithms for a Daring Escape

Heavy footsteps pounded the wooden deck above. Frank glanced around, assessing their limited options. The only hatch led up to the deck and the new arrivals. The hold itself was nearly empty, the crew having unloaded the cargo upon arriving in Usb. Trying to hide here would amount to standing in a corner and whispering "You can't see me."

As Frank listed and discarded every possible option, including the rarely effective ploy of lying down and playing dead, he saw Notation pull out her badge and stand at attention.

"What are you thinking?" he hissed.

"I am an officer of the law on an official investigation," explained Notation.

Frank shook his head in disbelief. "The 'Stop in the name of the law' routine isn't about to work here. Or most places, for that matter. We're on a smuggler's ship, investigating the theft of police property. No one on the force even knows you're here, do they? And I'd be willing to bet whoever is coming through that door knows that."

Notation opened her mouth to argue, but paused and closed it again. She slid her badge back into her jacket as a stream of large and surprisingly well-dressed thugs poured through the door. They

spread out in the hold and formed a loose circle around Frank and Notation.

"Ladies and gentlemen," said Frank. "We have concluded our inspection and it appears that you are not carrying any rotting eels. Thank you for your patience as we strive to ensure the safety of this kingdom's food supply. We'll be on our way now."

By way of reply, two of the larger thugs grabbed Frank's arms. Together they lifted him off the ground and proceeded to carry him back up onto the deck. Years of experience had prepared Frank for this reaction and he had developed a technique for positioning himself so as to minimize discomfort, but he could still almost feel the bruises forming under their powerful grip.

"Hey!" Notation's shout indicated that she was being similarly escorted outside.

Frank blinked as they emerged into the sunlight. The men carried him to the middle of the deck and dumped him on the wooden floor. Notation thudded next to him, and the thugs again formed a loose circle around them.

Frank slowly pushed himself into a sitting position and eyed their captors. They swayed with the ship but otherwise didn't move. They appeared to be waiting, which meant that whoever was in charge hadn't arrived yet. Frank seized the opportunity and turned to the nearest thug.

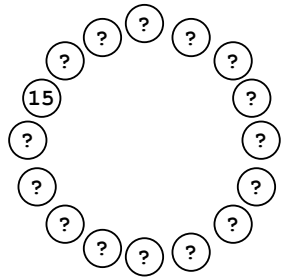
"What's the plan then?" asked Frank. "Lock us away? Drop us over the side? Hand us over to your boss's employer?"

The man shrugged. "Don't look at me, I've only been working here for fifteen days."

"A newbie, huh?" said Frank.

The Vinettees were fanatical about limiting information. They only shared their plans with the most senior person on the crew. New hires were required to prove their loyalty as they worked their way up the ranks. To get any useful information, Frank would need to find the most senior person aboard.

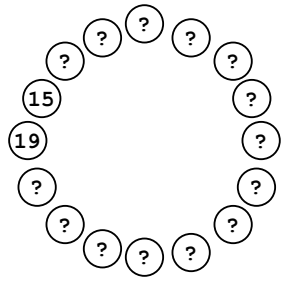
A plan started to form in Frank's mind. The Vinettees' crews always arranged themselves in order of seniority. It had something to do with mentorship—the most junior crew member served as a new hire's mentor and so forth up the ranks. In group situations, they all tended to stand next to their mentors.



The circle of thugs was nothing more than a sorted array that had been bent into a loop. Binary search would almost work here, but it would have to be adapted to fit the organization of the data: a circle, rather than a straight line of values. Unfortunately, this meant that Frank didn't know where the array started or ended. Quickly he developed a new algorithm to efficiently search for the most senior crew member. In this case, efficiency meant both minimizing the number of thugs with whom he would have to converse and maximizing his chances of getting answers before they caught on.

He turned to the woman on the thug's right. "How about you? Are you the veteran here then?"

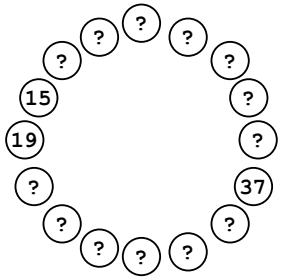
"Nineteen days," she said.



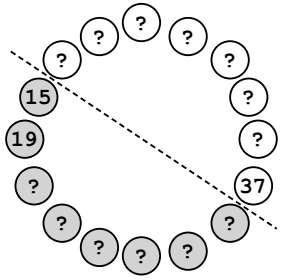
Now Frank expected that he had an ordering—that seniority increased as he went counterclockwise around the circle. But he couldn't be sure yet. As absurd as the possibility was, Fifteen-Days and Nineteen-Days could be the most junior and senior thugs, respectively. He had been careless with choosing how to start searches before; it never ended well. He need another data point. So he chose the middle of the remaining range of thugs.

"How about you?" he asked a woman across from Fifteen-Days.

"Thirty-seven days," replied the thug. "What's it to you?"



With this piece of information, his hunch of a counterclockwise ordering was confirmed. Therefore, he discarded everyone between Fifteen-Days and the person before Thirty-Seven-Days from consideration—the most senior thug, if it wasn't Thirty-Seven-Days, would have to be counterclockwise from her, but before Fifteen-Days.



Frank stifled a rising swell of annoyance. He had never faced such a junior crew before. He actually felt a little insulted. "This is getting embarrassing," Frank said to their captors. "We were caught by a bunch of newbies. What are you all, the backup team?"

"What are you doing?" whispered Notation.

"A modified binary search," Frank growled back.

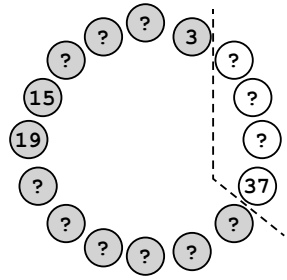
Notation sighed. "I figured that out. It looks like you're searching for the most senior one. You're not exactly being subtle. But why? And how do you know they're standing in order?"

Frank ignored her. He took a deep breath and refocused on the task at hand. He didn't know how much time he had before The Boss showed up. He chose the middle of the remaining range. "And you?"

"This is my third day," replied the man, hesitantly.

"Come on!" shouted Frank. "Really?"

"Three days? You're not in training or anything?" Notation asked, sounding genuinely curious.



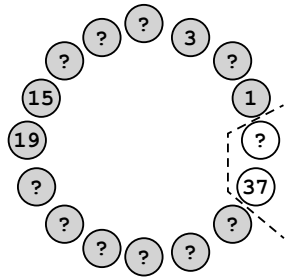
Frank refined his range again, accounting for the fact that the most senior person couldn't be Three-Days, Fifteen-Days, or anyone in between.

Frank divided the remaining range in half again. "You must be a relative veteran then, right?" Frank asked.

"Uh . . . this is my first day, sir," stammered the thug. He began sweating profusely as everyone's attention turned to him.

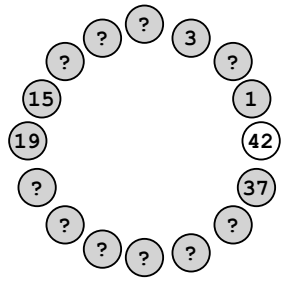
Frank cursed under his breath.

"Don't call him sir," shouted Nineteen-Days. "He's our prisoner."



Frank had now limited the search to a very small window. "And I suppose this is your first day too?" he asked the last thug under consideration. He didn't bother hiding his disdain.

The woman laughed. "I've been with the Vinettees for over a month," she said. "Forty-two days of keeping nosy cops out of the way."



Bingo. "Really?" said Frank. "What are you doing here then?"

The thug frowned. "What do you mean?"

"The Vinettees usually reserve their senior henchmen for more important tasks. Babysitting a shipment of smuggled cabbages seems like a waste of your time," he said, struggling to remember whether there had been anything about cabbages in the log. It was a reasonable bluff anyway. All smugglers dealt in cabbages at some point. The recent increase in cabbage taxes had practically doubled the black market trade in Usb.

"Cabbages?" the woman scoffed. "I did the cabbage route on my first day. We have more important work now."

"Really?" said Frank. "Move all the way up to carrots?"

The thug turned a bright shade of red. Even though they often accounted for over 80 percent of a smuggler's profit, somehow vegetable smuggling remained the embarrassing side of the business.

"No," she said. "A hundred times better than carrots. A private contract."

"Really?" said Frank. "I hear that fencing carrots is good business. Pays well, they say."

"Oh, don't worry about that," said the thug with an air of smugness. "The League pays us well for our service. We were told—"

"Mr. Runtime." A familiar voice cut through the air. "Stop trying to weasel information out of my employees. You will find that the information is no more useful than it is challenging to procure. They, of course, know nothing of value."

Frank looked up to see Rebecca Vinettee join the circle. His stomach clenched.

"And you are?" prompted Vinettee, looking at Officer Notation.

"This is Susan Pointer from the Bureau of Food Safety, Pickled Eel Division," said Frank. "We're looking into a bad shipment of Usb Greytails."

Rebecca Vinettee made a soft tsking sound. "No, Mr. Runtime. I don't think so." She paused, studying Notation carefully. "If I am not mistaken, this is Officer Elizabeth Notation. First year on the police force."

"I'm on an official—" started Notation.

"No," interrupted Rebecca Vinettee. "You are not on an official investigation, Officer Notation. I know all the officers currently assigned to official investigations, from grand theft aquatic all the way down to slug racing. My sources keep me well informed about such things, and you are not on any of their lists. But you *are* trespassing on my ship. So the question is what to do with you?"

"I thought the question was 'why?'" said Frank.

"Mr. Runtime," said Rebecca Vinettee with exaggerated patience. "Please do not underestimate me. I know the why. I knew the why before you did. I also know the who, when, what, and even the how.

"But the question remains, how should I deal with two nosy officers—oh, my apologies. You are an officer no longer, are you, Frank? I should ask: How should I deal with a nosy officer and a nosy disgraced former officer?"

Frank clenched his fists. His thoughts flashed back to his last month on the force and Rebecca's mocking laugh as she was released from the cells. He hadn't been able to build a case, at least not while following the book, and the captain had been forced to let her go.

"How about you bore us to death with a monologue?" asked Frank. "Why don't you tell us about the League?"

Rebecca laughed. "Don't worry, Mr. Runtime, I have no intention of keeping you around long enough to bore. I had, of course, already decided how to deal with you, if you ever chanced to get in my way again. The question was merely a kindness to give you the illusion of control over your own demise."

"Demise?" Notation squeaked.

Rebecca Vinettee nodded at the circle of henchmen, and they advanced, drawing an assortment of fancy weaponry from their expensive suits like magicians performing elaborate conjuring acts. One tall man produced a three-foot-long spiked club from under his tie; another thug slide a broadsword out of her sleeve.

Then, to absolutely everyone's surprise, a barrel landed on the deck with a loud *thwack* and burst open, spraying pickled eels across the planks.

POLICE ALGORITHMS 101: ADAPTING YOUR BINARY SEARCH

Excerpt from Professor Drecker's Lecture

Not every computational problem you face during your career is going to have a nice, prepackaged solution. Sure, scholars have spent years studying a vast range of problems and writing up solutions. But in the field you'll find new problems with new wrinkles. If you leave this class having just memorized a sheet of algorithms, you're soon going to find yourself in deep trouble.

In order to handle novel problems, it is important to understand how an algorithm works and how it can be adapted to new problems. The basic idea behind binary search—using the structure in the data to repeatedly halve the search space—is more important than the details of a particular application. Using that one piece of intuition, you can adapt binary search to search circular (but still sorted) arrays or even determine the optimal temperature for your coffee, by testing warmer and cooler coffee until it's "just right."

—8—
Socks: An Interlude and an Introduction

Chaos erupted on the deck as the thugs danced away from the wash of pickling juices and sliding eels. A second barrel landed a moment later, sending out a fresh wave. The eels, Deepwater Longbacks by the look of them, measured around four feet from end to end and threatened to trip and tangle, not to mention disgust. A group of three thugs fell into a heap as they tried to escape the splash zone.

Frank jumped to his feet, grabbed Notation's arm, and yanked her up.

"What's going on?" she shouted as a third barrel sailed over their heads and hit the railing.

"No idea," said Frank. "But now's our chance. This way!"

He pulled Notation toward the railing at the ship's edge, sliding through the puddles of pickling juices and piles of eels. A fourth barrel hit the deck and sent the thugs scurrying again. The deck was beginning to look like a grotesque bowl heaped with long, pale gray noodles.

Alongside the ship bobbed a small, nondescript, but oddly familiar schooner loaded with more barrels. A teenage boy, dressed in wizard's robes, was rolling a new barrel onto a contraption composed of planks.

"We should jump for it," said Notation.

"Jump," agreed Frank, eyeing the other ship. He didn't recognize the boy but figured that he must be an enemy of the Vinettees. Nobody in their right mind would fling pickled eels at the Vinettees unless they were already at war. Then again, very few people in their right mind would chose eels as a weapon under any circumstance.

Notation nodded, pulled herself onto the railing, and jumped without hesitation. She sailed through the air, clearing the gap easily, and landed on the schooner's deck.

Frank pulled himself up onto the railing, muttered a handful of Boolean curse words, and hesitated again as he balanced on the wooden beam. Then, after counting to three a few times, he jumped. Frank missed the schooner by two feet and plunged into the freezing water. He surfaced in time to see another barrel soar overhead. He paused briefly to wonder if the boy was still flinging eels or whether he had moved onto something else—soft cheeses, perhaps.

Notation reached down and pulled him onto the ship. He got to his feet and began wringing out his cloak while he assessed their

new situation. The teenager was continuing his barrage of the *Retry Loop*. Sailors hurried around the deck, pushing the smaller ship away from the *Retry Loop* with long wooden poles.

"Mavis?" Frank called loudly. "I know you're here somewhere."

Mavis appeared a second later from the hold. "He paid for the cargo," she said, pointing at the teenager. "It's up to him how he wants to unload it."

"I thought you couldn't—"

Mavis waved him off. "I owed the Vinettees one anyway," she explained. "Last month they stole one of my caches—17 bushels of carrots!"

Frank stared at her in amazement. Going against the Vinettees was dangerous business. A hundred bushels of carrots wouldn't be close to worth it.

"Don't look at me like that, Frank," she said. "I'm not doing this out of the goodness of my heart. It's a paid contract. Simple as that. Socks made it worth my while."

"Socks?" asked Frank.

"Socks?" asked Notation.

Mavis motioned toward the boy, then shrugged. "That's what he called himself. In my business, you don't press for real names."

"You didn't ask for official identification before letting him board?" asked Notation. Mavis gave her a skeptical look.

By now, Mavis's crew had gotten the *TCP Flyer* turned around, and they were heading out to sea. Dirty sails were hoisted into the air, blocking Socks's barrel throwing. He took a last long look at the *Retry Loop*, smiled, and turned finally to Frank.

"Hi," he said with all too much cheerfulness, "I'm Socks." He held out his hand. Frank shook it warily.

"Thank you for saving us, *Socks*," Notation said as she shook his hand too. She eyed him beadily, then, apparently unsatisfied with the subtlety of her approach, followed with, "What's your real name, Socks?"

"Unfortunately, that is my real name," replied Socks a little sadly. "My full name is Socks Repellent, Officer."

"Oh . . . " Notation trailed off. Evidently failing to find any suitable consolation for such a misfortune, she repeated, "Thank you for saving us."

"Anytime," replied Socks. "I didn't have any experience flinging barrels, so I'm glad it worked."

"Why did you help us, Socks?" Frank asked. "More importantly, how did you know we needed help?"

"Well," said Socks. "You see . . . I've been following you both all morning."

"The alley on Three Bit Lane?" asked Frank.

"Yes," replied Socks, blushing. "And behind the big ArrayCart in parking space #2."

"I missed that one," Frank admitted.

"His legs were clearly visible by the wheels," said Notation.

"Why were you following me?" asked Frank.

"Us," corrected Notation.

"Because we're after the same people," said Socks as though it were the most obvious answer in the world.

"We are?" asked Notation.

"I think so," said Socks, suddenly looking uncertain. "You're investigating one of the police department thefts, right? I saw Captain Donovan visit Frank last night, so I assumed it had something to do with the one at headquarters."

"*Thefts*?" asked Frank. "There's been more than one?"

"Oh," said Socks. "You didn't know. Sorry. Maybe I wasn't supposed to say anything. But I don't think that was supposed to be secret. I guess—"

"Why are the wizards concerned about thefts from police stations?" Frank cut in.

"The king called them in. A few weeks ago, King Fredrick summoned some of the kingdom's most senior and respected wizards to

investigate a theft. I'm Gretchen's apprentice, of course. It's only my second year, but—"

"Why did he call in senior wizards?" Frank interrupted again. Gretchen wasn't a name he'd heard before. Evidently, the more powerful wizards had been occupied with more pressing tasks, and the king had worked his way down the list, but Frank didn't want to embarrass the boy by questioning his mentor's seniority.

"The Capital Police are well equipped to handle thefts," added Notation. "It should be their investigation."

"The king called in the wizards to consult about the mask and whether they could find it," explained Socks.

"What mask?" asked Notation.

"You don't know about the mask?" said Socks, now a little panicked. "Oh, dear. Maybe I shouldn't have said anything."

"What mask?" growled Frank. He rubbed his temples and took a few deep breaths.

"Mr. Repellent," took up Notation in her official police voice, "we are on an important investigation. If you have information that would help in this case, it is your duty to provide it. Please tell us everything."

"Start at the beginning," added Frank.

It took 10 minutes for Socks to finish his excessively detailed story concerning a monthlong crime spree in which a multitude of unrelated items were stolen from secure locations. The most disturbing revelation involved the theft of a dangerous magical artifact, an enchanted mask, from a military convoy. When pressed for details on what the mask did or why it needed an armed escort, Socks would only repeat that it was "extremely powerful." The awe in his voice made Frank nervous.

"You think this theft is related to the one at the police station?" asked Notation.

"Gretchen thinks so," Socks answered. "In every case, the guards were completely unaware that a theft had occurred until

the next day. She thinks the thief used a memory spell or a sleep charm."

"I did not fall asleep," Notation said with such force that even Frank leaned away.

"I . . . I didn't mean," Socks stammered.

Frank left him to flail about for words as he pondered the kid's story. Something about it didn't add up. "Why tail us?" he asked finally. "You're on an investigation sanctioned by the king himself, right? You could have just walked up and introduced yourself."

"Yes! But . . . " Socks trailed off.

"But?" asked Frank.

"I wasn't sure it would be worthwhile," admitted Socks.

Frank glared at him until he continued.

"I wasn't sure you were going to find anything interesting," explained Socks. "Once I introduced myself, I would be obliged to come with you and help out, right? I wanted to keep my options open. What if I heard about a better, magical lead or something? Sorry," the boy muttered.

They lapsed into thoughtful silence as the ship's crew continued to dash around them, doing whatever was needed to make a ship move. As far as Frank could tell, it seemed to primarily involve pulling on ropes.

"So . . . what now?" asked Socks.

"We follow leads," said Frank.

"We found two suspicious ports in the *Retry Loop*'s logs," Notation explained. "Our next step is to investigate them, starting with the Port of Mudwall. In fact, I should inform the captain of the ship. Of course, we'll need to borrow her vessel for the investigation."

Frank laughed. "Good luck explaining that to Mavis."

"No need," said Socks hurriedly. "I already paid to contract the ship. You just need to tell Mavis where to sail."

"Excellent," said Notation.

"Which means you're coming with us, doesn't it?" asked Frank, without trying to conceal the irritation in his voice.

"Of course," said Socks. "I paid for the ship. And I saved you. And I seem to know more about the case than anyone else here. And I'm an apprentice wizard. That could come in handy." His desperation seemed to increase with every additional reason.

"Mr. Repellent, we would be grateful for your assistance," Notation assured him. "Wouldn't we, Frank?"

"Yeah. Great," mumbled Frank. He had collected yet another tag-along whom he didn't trust. At this rate, he would have a boatful by nightfall.

— 9 —
Backtracking to Keep the Search Going

The Port of Mudwall didn't even live up to its unimpressive name. It wasn't much of a port; the rickety wooden dock could fit no more than two modestly sized ships. The mudwall itself, in theory a 20-foot-tall earthen city wall, had never actually been constructed, and was only hinted at here and there with 2-foot-tall humps surrounding about a third of the city.

Frank stepped over the dirt hump and made his way to the sole shop in the city, with Notation and Socks trailing along behind him.

As they entered the store, the shopkeeper's face transformed from surprise to outright delight with the progression of a powerful sneeze. He knocked a stack of carrot-themed tourist pamphlets off the counter as he hurried around to greet them.

"Hello!" he nearly shouted. "Welcome to Mudwall, home of the famous mud carrot farms. What can I get for you? Food? Supplies? Carrots? Carrot-flavored baked goods? We have a simply wonderful carrot pie."

"Information," said Frank.

The shopkeeper's face dropped. "Oh," he said. "You're not here for the Carrot Festival, then?"

"Carrot Festival?" asked Notation.

The shopkeeper nodded. "It's the 50th annual Carrot Festival later this week."

"You get a lot of visitors for that?" she asked.

"Not recently," admitted the shopkeeper. "Mudwall isn't the tourist draw that it used to be. Not since G'Raph started hosting its Mud Radish Festival. People would rather go there for the big-city atmosphere."

Notation and Socks looked at each other. Socks mouthed, "What's a mud radish?" Notation shrugged.

"What do you know about a ship that came through here a few days ago?" Frank asked.

"What ship?" asked the shopkeeper. "We haven't had a ship here in a few months. We've had a few donkey carts pass through on the coastal road. But no ships."

"Are you sure?" asked Notation. "Because we are on an official investigation, and it is very important that you tell us anything you might know about all ships that have come through here recently."

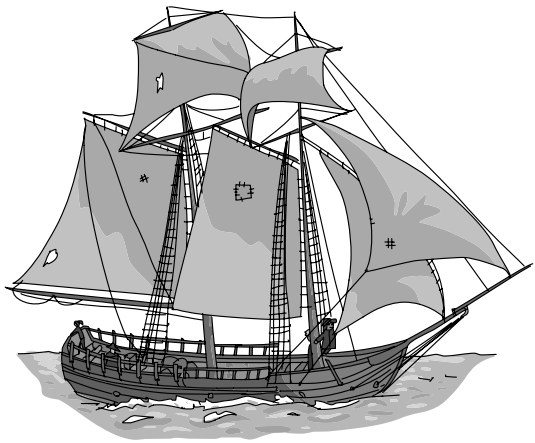
"I would have known if there was a ship," said the shopkeeper. "I have a wonderful view of the dock from my window. Even if I hadn't seen it, I would have heard about it. We don't get many ships these days. People would've been all excited. The Sound of Carrots, our three-person marching band, usually greets each ship. I certainly would have heard them."

Notation looked as though she was preparing for a new round of questions, but Frank cut in first. "Thank you, sir," he said. "We appreciate your time."

He herded Notation and Socks from the building and back into the muddy street.

"Do you think he was telling the truth?" asked Notation.

Frank nodded, scanning the street. "He seemed genuinely surprised at the mention of a ship," he said. "But it wouldn't hurt to ask around a bit more."



They interviewed a dozen of the town's residents, a sample that made up approximately half the population, before admitting defeat. No one had seen a ship. No one had heard of anyone seeing a ship. And no one could even understand why a ship would visit. The port clearly didn't receive much traffic.

"Maybe it came at night," mused Notation as they walked the dock toward the *TCP Flyer*. "They sent a small party ashore on a rowboat, handed off the documents, and left. Someone could have been waiting with a cart right here." She pointed to an arbitrary plank on the dock, which didn't look any more or less suspicious than its neighbors.

"Maybe," said Frank. "It doesn't matter, though. Without a witness, there's no trail."

"What does that mean? We can't find them? The investigation is over?" asked Socks.

Frank grunted a laugh. "No, kid. Dead ends are part of investigation work. That's why we use a backtracking search."

Socks stared blankly back at Frank. When comprehension failed to dawn on the boy's face, Frank added, "We keep exploring along the most promising direction until that trail hits a dead end. Then we backtrack to a previously unexplored, but promising, lead and search from there."

"So you have other leads?" asked Socks.

"A few," admitted Frank.

"If we backtrack to the next unexplored clue," Notation mused, "we return to the logbook. There was another destination listed: Frayed Cable Island."

Frank nodded and considered which leads remained. The senior thug had mentioned a League of something. That meant Frank's pitiful list of unexplored leads currently consisted of:

> Frayed Cable Island
> Threads from the ArrayCart
> Vinettees
> League?

The search would now backtrack to the logbook and the unexplored island. If they failed to pick up a trail on Frayed Cable Island, he would be forced to follow one of their more tenuous leads. The colored threads from the cart were unique enough to be moderately promising.

"So we aren't done yet?" Socks confirmed for the tenth time as the *TCP Flyer* traveled to Frayed Cable Island. "You still have a few leads, right?"

"Investigations backtrack all the time," Notation assured him yet again. "Don't you ever have to backtrack in your work? What about while making a potion?"

Socks looked shocked. "How would you backtrack while mixing a potion? You can't unadd spider legs or unstir the mixture."

"I meant backtrack while *developing* a potion. You try adding spider legs and realize that it destroys the magical consistency or something? So you cross out that instruction and try a different approach the next time you mix the potion. Each recipe you try is a

single state in your search space, and you backtrack by returning to the last good recipe."

"Oh," said Socks. "You mean *revising*. You have to revise spells and potions as you develop them. You're still moving toward your goal, just not in a straight line. Nobody gets it right the first time."

"Exactly. It's the same thing," said Notation.

Mavis joined the group, adding, "It's like when you're searching a cave for a 'forgotten' stash of goods, you explore down different paths and backtrack when you don't see anything of value."

"Yes," Notation agreed, hestitantly. "Backtracking search is much like exploring caves. Although I have to point out that it would be difficult to tell if an object found was forgotten or not. I guess you could—"

"And speaking of searches, we're at your next destination," interrupted Mavis. "Frayed Cable Island doesn't have any docks. We'll anchor the *TCP Flyer* here, but you lot will have to row in the rest of the way."

POLICE ALGORITHMS 101: BACKTRACKING
Excerpt from Professor Drecker's Lecture

Almost every investigation will involve some backtracking. Even the best officers can't always follow clues in a straight path from start to finish. The world is filled with useless information, ambigious clues, and red herrings. On top of that, you'll make mistakes. So it's important to know how to backtrack a search when you hit a dead end. Simply put, this means backing up your search to a previous state and trying a different option.

continued

Up until this point, algorithms have been presented on search spaces where they can efficiently jump from any state to any other arbitrary state. For example, in an array, you can easily examine the value at any location using just its index. And in a hotel hallway, you can run between rooms. This flexibility provides efficiency for the algorithms.

However, many search spaces come with constraints on how you can move from state to state. If you are searching a castle in the physical world, you can't jump arbitrarily between rooms—you have to traverse the halls and rooms in between. Similar constraints can apply in the computational realm for data structures like graphs or linked lists.

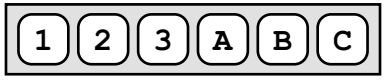
Even in cases where you can jump between states, it is often useful to picture backtracking as moving through previous states on your way to explore new states. In the algorithmic world, this can be significantly less costly than physically retracing your route. However, the processes are conceptually similar: you back up the search and proceed down a different avenue.

In the following lectures, we'll see many examples of searches that backtrack when they hit dead ends. And once you join the force, you'll experience more dead ends than you ever imagined.

— 10 —
Picking Locks with Breadth-First Search

Frank, Socks, and Officer Notation huddled by the back gate of the prison's outer wall. Despite its truly impressive coating of rust, the locked gate had resisted both of Frank's attempts to kick it open. He had only succeeded in clouding the air with red dust and introducing Notation to at least six new Boolean curse words.

"So . . . that didn't work," supplied Socks. Frank ignored him and studied the locking mechanism. It was a standard carved keypad with buttons labeled 1, 2, 3, A, B, and C in a single ordered row and an ENTER button beneath.



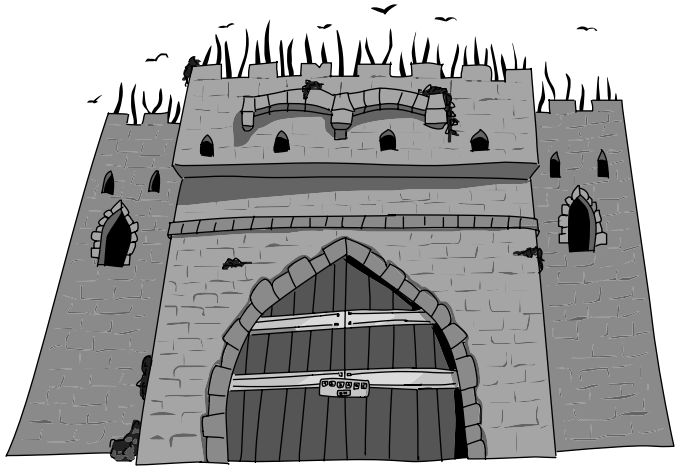
"We'll have to do this the old-fashioned way," said Frank.

"Wasn't kicking down the gate the old-fashioned way?" asked Notation.

Frank ignored her as well. "Socks, do you know any magic lock-picking spells?"

"No," Socks protested loudly. "Those are illegal!"

"How about something to weaken the lock? Or maybe the hinges?" asked Frank.



"You want me to help you destroy property?" Socks looked aghast. "That's worse than lock picking. Do you know how much trouble—"

"Search spells, then? The Spell of All Combinations or the Spell of Breadth-First Search?" Notation interrupted. She'd heard enough on the topic of proper and improper spellwork after Frank had casually inquired about the feasibility of replication spells on gold coins—a use of magic that fell firmly on the wrong side of both Socks's and her own ethical line.

"I've used the Spell of Breadth-First Search a few times," Socks answered. "My real expertise is binary search trees, but I'm familiar with a range of computational techniques. Once I—"

"Will breadth-first search work on the lock?" interrupted Frank. Over the years, Frank had worked cases with a handful of wizards of varying levels of respectability. He'd seen at least a dozen different lock-picking spells but had never seen a door opened with an explicit breadth-first search.

Notation smiled. "Definitely! It's a bit abstract, but I saw a similar problem recently in my Police Algorithms course. When you think about it, a code lock is just a search problem; you enter a string of characters to open it. The search space is all possible strings that can be made from those characters. Every string is a

valid search option, from a single character like 1 or A to complex sequences like ABC123CBA321. The search target is the one string that opens the lock."

"But we don't even know how many characters we need," protested Socks. "The lock could have a 30-character combination."

"That's why she suggested breadth-first search," said Frank, thinking aloud as much as addressing Socks's concern.

"I don't understand," said Socks.

Notation quickly picked up the explanation. "You see, breadth-first search expands outward from a starting point, exploring along a frontier of solutions. Naturally it will try the shorter solutions first."

"Huh?" asked Socks, now looking confused to the point of panic. "I thought breadth-first search used magic lists. I've always used a magic list. Isn't it just a magic list?"

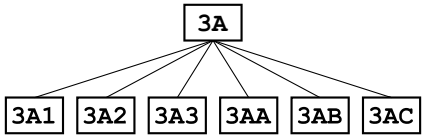
"Yes," agreed Notation. "Breadth-first search maintains a list of options to try next if the current option doesn't work. The algorithm is basically a loop that keeps pulling options from the front of the list and adding new options to the back. On each iteration, we pick a new option to try from the *front* of the list. And, if that's not what we want, we check if there are any new options reachable from the current one and add those unexplored options to the *back* of the list.

"You start at a single point in the search space, in this case at a password of length zero. Then for each password you try, you add new search possibilities to the end of the list. In this case, each time we try a password, we'll add all single-character extensions to the list. For example, here we know the password can only contain the characters 1, 2, 3 and A, B, and C. Once we've tested 3A, we'll add 3A1, 3A2, 3A3, 3AA, 3AB, and 3AC to the end of our list."

Socks screwed his face up in concentration, then asked, "How do we know which options to add?"

"Think of it like a tree of possibilities," suggested Notation. "Each branch, or *node*, is a password from our list, like 3A. The neighboring options are the nodes under it—the passwords we would get by

adding one more character to the end. Breadth-first search progresses down each level of the tree before moving onto the next."



"Since we add the new, longer passwords to the *end* of our list, we try all the short stuff first," Frank threw in. "Now, can you do it?"

"This isn't a proper use of—"

"Come on! Really?" interrupted Frank.

"It's basically a lock-picking spell," responded Socks.

"Yes. That's exactly what it is!" shouted Frank.

"Forget it," said Notation, throwing her arms up in frustration. "If he doesn't feel comfortable picking the lock, we're not going to change his mind by yelling." She turned and studied the stone wall itself, which stood at least 10 feet tall. After a moment she continued, "Frank, if you give me a boost, maybe I can climb over."

Frank gave the wall a skeptical look. Despite having been abandoned for years, the wall lacked the large cracks and rambling vines that often aid mountaineering efforts on old castle walls. The workmanship was impressive. Someone had taken real pride in building this wall; you could tell from the artistic way the metal spikes twisted as they jutted up. Those little details took effort.

"Maybe. It's pretty high, though, and those spikes look awfully sharp," he said.

"It'll be just like the obstacle course at the academy," said Notation. "Aside from the hard-packed ground, the lack of handholds, and the large metal spikes, that is."

"Those'll probably add some excitement," Frank offered.

"Shut up and give me a boost, Frank."

"No. No. I'll do it," said Socks hurriedly. "I'll use the Spell of Breadth-First Search. I'll need something for the list, though. A roll of parchment, perhaps?"

Frank and Notation looked at each other. "No can do, kid. Use the ground; it's muddy enough."

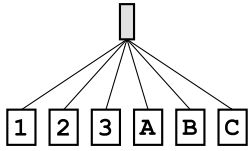
"Oh. Yes. Of course."

A few minutes later, the lock began to glow. "Here we go," said Socks.

The word ENTER glowed briefly, followed by a clicking noise. But the gate remained locked. The spell had tried the first password, which was nothing at all. Next a series of numbers and letters appeared in the mud:

1, 2, 3, A, B, C

Frank could picture the tree of possible passwords that the list represented.

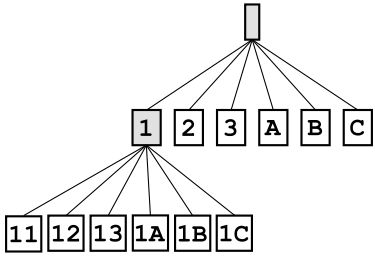


An instant later, the number 1 glowed, followed by ENTER. Again the gate clicked, but didn't open. The list on the ground changed, showing the new list of passwords to try, branching out to the third level of the tree.

2, 3, A, B, C

11, 12, 13, 1A, 1B, 1C

But these were added to the end of the list. The search itself continued on the current level, trying 2.



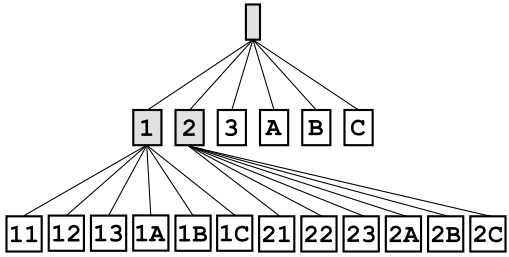
The password 2 didn't work, and the list grew again.

3, A, B, C

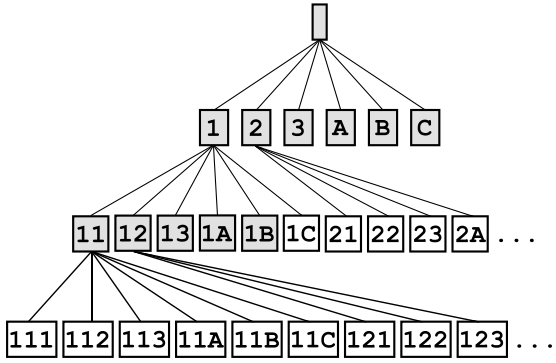
11, 12, 13, 1A, 1B, 1C

21, 22, 23, 2A, 2B, 2C

Again the tree branched out with new possibilities, but the search still worked its way along the current level, trying all one-character passwords before moving deeper.



In other words, the search explored the full breadth of each level before moving on to deeper levels.



The search finished the first level, trying the passwords 3, A, B, and C, before Socks broke the silence. "This could take a while."

Frank nodded, eyes fixed on the ever-growing list of numbers. "Notation, why don't you scout around the front?"

"Okay," she agreed, her expression betraying great relief. Rookies didn't tend to handle stakeouts well. Sitting still for hours on end with nothing to do wasn't something you could teach at the academy, although Professor Cloud's Philosophy of Law Enforcement lectures came close.

Five minutes after Notation left, the lock gave a loud click, and the gate swung open noisily on its well-rusted hinges. The list in the mud faded as the search algorithm completed.

"1111," said Frank, without a trace of surprise. It often paid to keep the codes simple enough for the henchmen to remember.

He used a stick to write the code in a patch of mud and circled it twice. Even a rookie couldn't miss the message. Then he turned to Socks. "Let's go."

POLICE ALGORITHMS 101: BREADTH-FIRST SEARCH

Excerpt from Professor Drecker's Lecture

Breadth-first search is an algorithm that explores search states in the order in which they are encountered. In other words, it always attempts to explore the oldest unsearched state first.

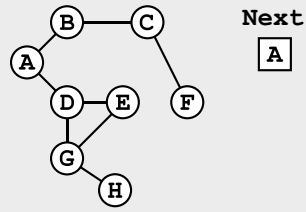
You can visualize breadth-first search as keeping a list (or, more formally, a *queue*) of known but unexplored states. At each step, the algorithm picks the next state to explore from the front of the queue. As the algorithm discovers new options, it adds them to the back of the queue, to make sure all previous options are explored before it moves on to new options.

It's helpful to describe breadth-first search in terms of how it explores a graph. A graph is a data structure composed of

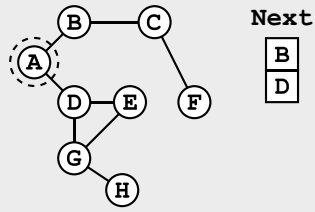
continued

individual *nodes*, with *edges* linking those nodes. If two nodes are connected by an edge, we say they are *neighbors*, which means you can move between those nodes. During your orientation, you studied at least one graph—the Kingdom Highway Map. This map represents each city as a node and the highways connecting them as edges. Make sure you own a good copy of that map. Criminals have a tendency to flee the city, and you'll need to know to which neighboring cities they are most likely to go.

Searching the Kingdom Highway Map is a classic graph search problem. Our search states are the nodes of the graph—the cities on the map. Imagine that a crime has occurred in city A and it is your job to find the fleeing criminal.

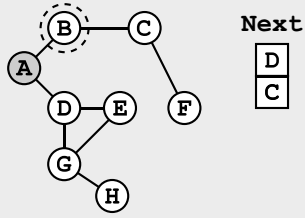


Breadth-first search explores along an expanding frontier, checking each node X steps away from the initial node before proceeding to any nodes $X + 1$ steps away. After you explore city A, its two neighbors, B and D, are added to the back of the queue. No other cities were in the queue, so B is the next city you'll visit.

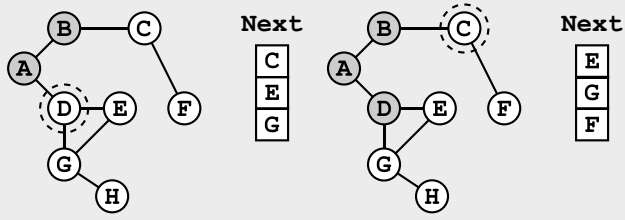


If each node has many neighbors, maintaining the queue of nodes to explore can use a large amount of memory. This memory requirement can become expensive in large search problems. As an officer, you'll want to invest in a number of good notebooks.

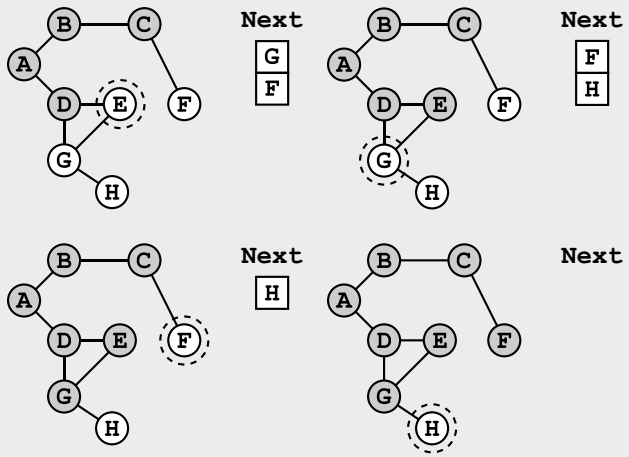
At each step in breadth-first search, we test whether the current node is the target node. In this example, that means thoroughly checking the city for our criminal. If the current node isn't the target node, we add only its previously unseen neighbors to the list. (A node that is *unseen* hasn't been added to the list yet.) We thus avoid adding either nodes that we have already explored or unexplored nodes that are already on our list. In this case, for instance, after checking city B, we would not add A to our list again.



Note that checking whether a neighbor is unseen requires even more memory because we must keep track of previously seen nodes. However, the benefit is significant—we avoid loops through previously explored nodes. Again, carefully keeping track of your search can pay off significantly.

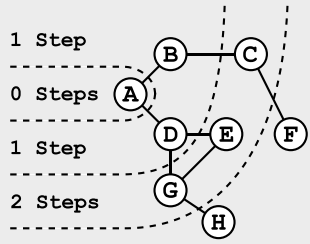


continued



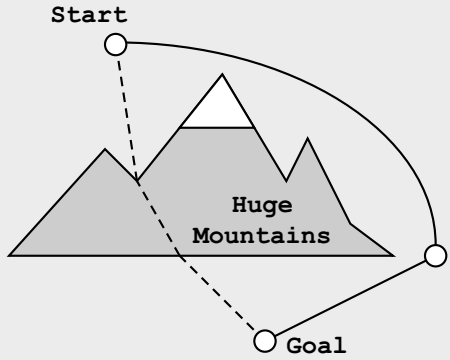
In this particular example, we find our suspect hiding in city H. We can stop our search there and make the arrest.

In search problems where moving between any two neighboring nodes has the same cost (time, energy, etc.), breadth-first search is guaranteed to find a path with the least total cost. It accomplishes this by expanding outward from the starting node, exploring *every* node that is X steps away before exploring any state that is $X + 1$ steps away.



Breadth-first search can even be adapted to return the shortest path by keeping *back pointers*. Each node keeps track of the node that preceded it. Then, upon finding the goal state, you can trace the pointers backward to re-create the path.

However, keep in mind that this works only if each move between neighbors has the same cost. In the general case, minimizing the number of steps in the search space can be very different from minimizing the cost of the path to the goal. For example, if hikers want to minimize their energy expended (cost), they would prefer a longer route that avoids crossing a mountain range. While the mountain pass would be shorter, and arguably more scenic, it could require significantly more energy.



— 11 —
Depth-First Search in an Abandoned Prison

Two steps into the prison, and Frank knew they had walked into a maze. The old computational prisons used to rely on their bizarre structure as much as on guards. Potential escapees think twice about sneaking through a door when they don't know what lies on the other side: freedom or the guard's breakroom.

"How about some light?" suggested Frank.

"Oh. Right," agreed Socks. He muttered an incantation and a bluish flame flickered from the end of his staff, lighting up the completely unremarkable room.

The square room, rough stone walls, and heavy oaken door were enough to confirm what Frank already knew: the entire structure was a grid of rooms, each with doors to only some of its neighbors. They would have to navigate from room to room. But since they didn't know which rooms had doors between them, they would have to search out a path as they went.

"Time for another search," he said.

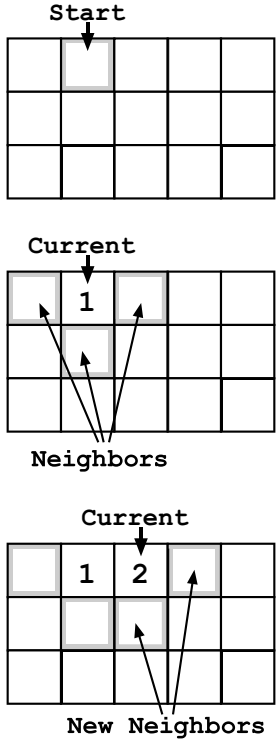
"A search?" asked Socks. "For what?"

"The papers, of course," responded Frank. He had no doubt that the papers were stashed here. An abandoned prison provided an

ideal location for stashing stolen goods, clearly surpassing the more commonly used warehouse. Arguably, the only better location would be an abandoned castle—provided it had a moat. The question now was whether they could find the documents and then, if they did, whether the documents would provide any valuable clues.

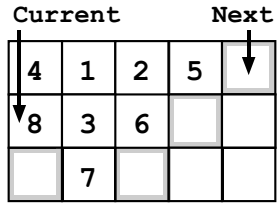
"Not another breadth-first search," protested Socks.

Frank considered the idea. In theory, breadth-first worked fine on a grid. Each state of the search space was a grid square. Once you explored one grid square, you could add its unexplored neighboring squares to your list of things to try. Frank could clearly picture the search propagating out over an empty grid, like a wave moving across the water.



However, breadth-first search had one major drawback in the physical world—an excessive amount of backtracking. Since you

were always adding items to the end of the list, the next square to explore could be annoyingly far away. Even on an empty grid, without walls blocking your path, you could find yourself hiking back to the other end of the search space.



It was the type of unnecessary movement that Frank made it a policy to avoid.

"No," said Frank. "Too much backtracking. We're better off going depth-first here."

"Depth-first search. Depth-first search," Socks mumbled to himself as though willing the spell into his memory. "I—I don't think I remember—"

Frank waved him off, and strode confidently down the corridor. "We don't need a spell for this one. I've been doing depth-first searches through buildings since you were in diapers."

"No backtracking with depth-first search then?" asked Socks.

"There's backtracking with most search algorithms. But backtracking in a depth-first search is better suited for walking."

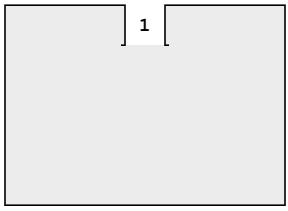
"Um . . . I see."

"No, you don't," said Frank bluntly. "If you don't know the algorithm, just ask. Pretending to know algorithms is a recipe for disaster. I've seen too many rookies tripped up due to bad searches. Good kids, like you."

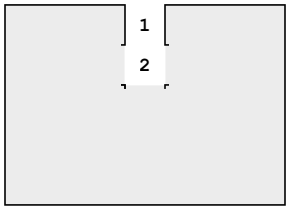
"Okay. What is depth-first search?" Socks asked.

"It's a simple algorithm," explained Frank. "Basically we explore deeply down each path. We go down one path until we hit a dead end. Then we backtrack to the most recent path that we didn't take and try that. We'll stop when we find the target.

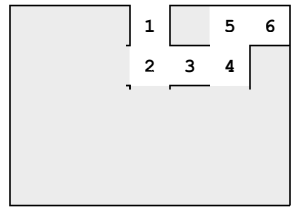
"In this case, we're going to use clockwise ordering. Whenever we have multiple options, we'll try north, east, south, then west—avoiding paths we've already tried, of course. We'll use the same ordering at every intersection, so we'll always prefer going north if we can. But in this case we have only one option, so we start by going south."



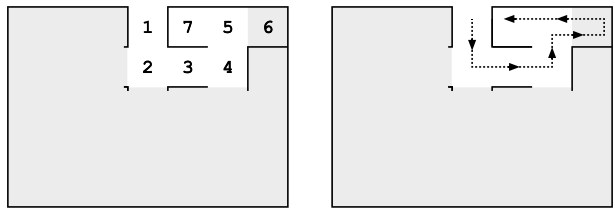
Even as Frank spoke, they reached their first decision point. Frank surveyed the options. They had come from the north, so he chose east—the next unexplored direction in his ordering. Before leaving the intersection, he retrieved a piece of chalk from his pocket and made a small mark on the wall.



After two more intersections—turning north, then east—they reached their first dead end. So far the rooms had either been completely empty or contained only the odd prison cell—the cells being enclosures within the rooms. With the complete lack of other distinguishing characteristics, Frank chalked a number onto a wall in each room and linked that number in his mind to the different mold formations he found there.

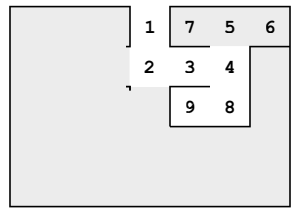


"Now we backtrack to the last room, room 5, with the mold that looked like a horse," explained Frank as they retraced their steps. This time they chose the only unexplored option from room 5, heading west. Unfortunately, they immediately hit another dead end—an empty room that sported a complex floral pattern of green and blue fuzz.



They backtracked through the most recent intersection whose options had been exhausted until they had a new option at room 4. The eastern option was a dead end, and they'd already explored the northern option, so this time they went south.

They ventured through two new empty rooms (8 and 9), differentiated only by the occurrence of a large stalactite of orange mold, which they stayed as far away from as possible. Orange mold was not known for its structural stability. After hitting another dead end, they found themselves retracing their steps all the way back to the first intersection in room 2.



"What if we miss it?" asked Socks in his now-standard worried tone. "Or what if we end up in a loop? We could be stuck forever!"

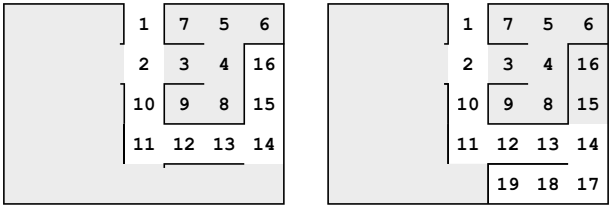
Frank groaned. "Listen, kid. This isn't my first time depth-first searching. I know what I'm doing."

"But loops."

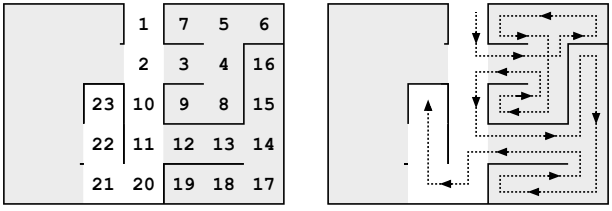
"Why do you think I'm marking the walls?" asked Frank. "If we avoid taking passages that we've already explored, we avoid going in loops."

Frank had learned that lesson during a Police Algorithms exercise. With the whole class watching, Frank had done six loops of the hedge maze before he heard another student loudly joke, "There he goes again."

They explored deeper into the maze, following snaking paths and backtracking at dead ends.



Then, in room 23, they found a small cell packed high with rolls of parchment and stacks of ledgers.



"We found it!" said Socks enthusiastically. His staff's flame cast a flickering blue glow through the room.

Frank felt the hairs on his neck rise as he took in the scene. He compared the height of the stacks with the mountains of paperwork he had completed through the years and did some quick calculations. The captain had never been shy about dumping paperwork on him, but Frank had still never seen anything like this. There were even mold-stained pages at the bottom of the stacks. Everything felt wrong.

Frank walked to the nearest stack and pulled off a sheet of parchment: a notice on the proper use of duck fences. The date and station number marked it as belonging to the stolen files. The next sheet, listing noise complaints in the Port of West Serial, also came from the stolen collection. It appeared equally random and unhelpful.

He knelt down and pried open a gap near the bottom, yanking a ledger free. The pages were spotted with a trio of mold-butterflies, but Frank could clearly make out supply lists for the castle guards. This ledger could have come only from the castle itself. He grabbed another book and found castle guard rotations for last November.

"This is wrong," he muttered. "There's too much here. There's castle ledgers as well." Frank shifted to an adjacent pile, starting again at the top.

"Is there a pattern?" asked Socks, as though he had just noticed the extent of the document piles.

"I—" started Frank, but he pulled up short as he opened another ledger, entitled *Transfer Requests*. Four pages had been torn from the middle of the ledger.

"Very strange," said Frank, flipping through the undamaged pages. "This could be—"

Frank was cut off as Socks stumbled toward him, flailing for balance. Behind him, Frank could see motion in the gloom. It wasn't until he heard the rusty shriek of the door's hinges that he realized what was happening.

"Door!" Frank yelled as the junior wizard fell into him.

The two of them tumbled to the ground. The door slammed. A loud click sounded as the lock engaged. Socks's staff, which had been dropped in the commotion, spun lazily into a tall stack of dry parchment. The staff's blue flame seemed much larger than Frank remembered.

Frank lay stunned on the stone floor as he watched the papers ignite.

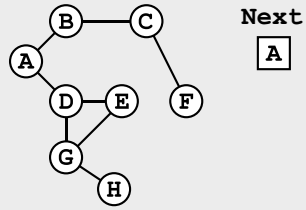
POLICE ALGORITHMS 101: DEPTH-FIRST SEARCH

Excerpt from Professor Drecker's Lecture

Unlike breadth-first search, depth-first search is an algorithm that explores more recently encountered search states first. The algorithm progresses down paths until it hits either the target or a dead end.

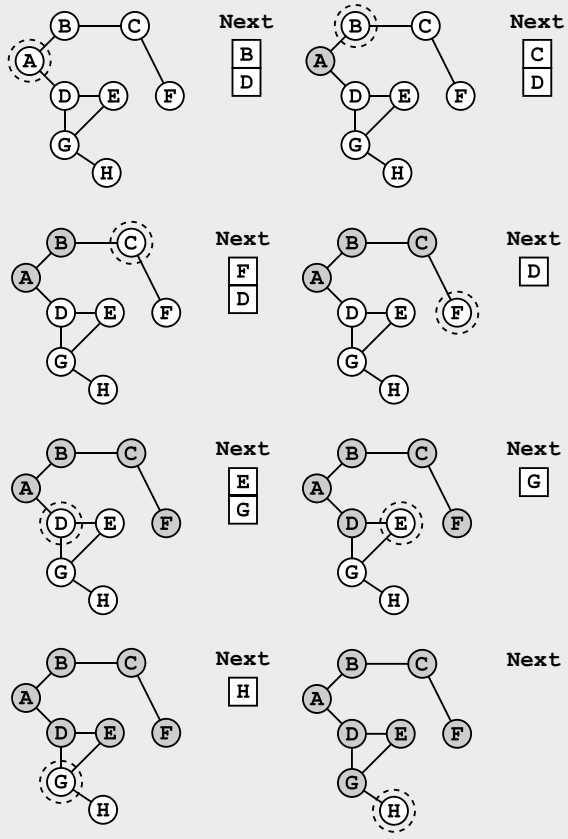
As with breadth-first search, you can visualize depth-first search as keeping a list (in this case, a *stack*) of known but unexplored states. At each step, the algorithm picks the next state to explore from the top of the stack. But unlike breadth-first search, depth-first search adds new options to the *top* of the stack.

Consider our graph example from the lecture on breadth-first search. Remember, graphs are data structures composed of individual nodes and edges linking those nodes. They can be used to represent all sorts of concepts, like city maps, networks of criminals, or even the layout of a castle. We'll use the Kingdom Highway Map from the same lecture and start our search from city A—the scene of the crime.



Depth-first search explores down one path until it hits a dead end (or a node it has already explored). In this way, the algorithm prioritizes exploring *deeply* down paths over exploring *broadly* over the options, as in breadth-first search.

continued



Once again, we find our suspect hiding in city H—although this time we travel a different path during our search.

As with breadth-first search, we avoid exploring nodes more than once by keeping track of previously visited nodes. This check is particularly important if you want to avoid falling into endless loops, checking the same nodes over and over again. In the above example, we avoid adding previously seen nodes (either explored or unexplored) to our list altogether.

— 12 —
Cafeteria Stacks and Queues

Frank pushed himself into a crouch and hurried to the door. He tugged and pulled, rattling and thumping the door against its lock. He grasped the rusted iron bars and threw his full weight into the effort, but succeeded only in producing louder clanking sounds.

Frank turned to Socks, hoping the young wizard knew a bar-bending spell. Given the circumstances, he felt confident that Socks would even consent to using a lock-picking spell. But as Frank's eye caught the smoldering stacks of parchment and the trails of smoke wisping to the ceiling, he froze. An image of a smoke-filled kitchen flashed across his mind, dredging up forgotten memories of his first year in the academy. He could almost hear the cook shouting. Frank shut his eyes hard, trying to force the memory away.

During his first two months at the academy, Frank had balanced his classes with a work-study job in the school cafeteria. The job wasn't anything glamorous; they didn't let new arrivals wash dishes, let alone prepare the food. Instead, Frank spent 15 hours a week transporting loads of clean trays, plates, and cutlery from the kitchen to the appropriate locations in the cafeteria.

Despite the tedious nature of the work, Frank found himself enjoying it. "Look at me! I'm undoing your work. I'm the anti-busboy," he would shout to the trio of busboys clearing tables and filling bins with dirty dishes. He unsuccessfully tried to break the school record for the most number of dishes transported in two minutes. He created an entirely new cafeteria game called Fling the Spoon. But it wasn't until a fortuitous run-in with Professor Heappens that he actually learned something from the job.

"Ugh. There are some data structures that just don't belong in the cafeteria," Professor Heappens muttered loudly as he studied the food options.

At 2:30 in the afternoon, the lunch rush had vanished, and Frank Runtime was hard at work transporting a load of bowls to the soup station. Though the comment wasn't directed at him, he found himself asking the professor, "What data structures?"

"Stacks," Professor Heappens said, looking up at Frank. "Stacks almost never belong in a cafeteria."

"Sure they do," Frank replied with the level of certainty that only new students and the truly ignorant can muster. He nodded down

at the stack of bowls he was carrying. “Stacks of bowls. Stacks of plates. Stacks of pancakes.”

Professor Heappens made a dismissive gesture and started walking away. “What do you know about data structures anyway?”

“How else are you supposed to arrange plates?” Frank asked. “If you laid them out end to end, they would take too much room.”

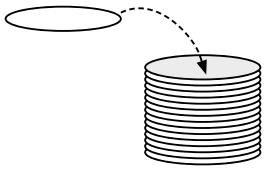
The professor stopped and stared at Frank with an expression of profound concern. After nearly a minute, he asked, “Do you know the difference between a stack and a queue?”

Frank shook his head. He hadn’t taken Police Data Structures yet.

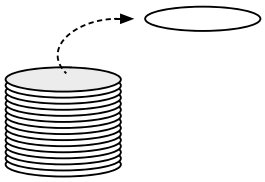
“A stack is a *last-in, first-out* data structure,” explained the professor. “It has two operations. You can *push* something onto the top of the stack. Or you can *pop* something off the top of the stack.

He gestured at the stack of plates waiting at the front of the line. “It’s just like the stack of plates over there. You can push a plate onto the stack.”

He placed his empty plate on top of the stack.



“Or you can pop a plate off the top.” He grabbed his plate back.



“And whenever you pop something off a stack, you get the newest item on the stack. The oldest item will stay at the bottom of the stack until you have popped off everything above it.”

“So?” Frank asked. “What’s wrong with that?”

"Nothing's wrong with a last-in, first-out data structure if you use it correctly. Stacks are wonderful if you are writing a depth-first search; you just keep pushing new search options onto the stack and popping them off when you backtrack. But cafeterias have misused stacks for decades!

"Take this stack of plates right here. Do you know how long the bottom plate has been there?"

Frank tried to recall the last time he had seen the stack empty, but couldn't even conjure the image.

"Five years!" shouted Professor Heappens. "I know, because I marked it. For five years that bottom plate has sat there unused, while students like yourself dump other clean plates on top. It sits there collecting dust around the edges.

"But that isn't even the worst. Look at what they are doing to the mashed potatoes!"

Frank glanced over at the large wooden bowl of mashed potatoes. A cook was in the process of refilling it. He held a large pot in one hand and was gleefully ladling fresh mashed potatoes into the bowl. It took a moment for Frank to realize the older food was simply being buried. His stomach turned.

"How long?" he croaked, not really wanting to know the answer.

"Don't worry. They wash out the serving bowl at least once a week, so the old mashed potatoes are less than a week old."

Frank didn't feel reassured. In fact, he felt rather ill. A quick scan of the cafeteria showed the last-in, first-out pattern being utilized everywhere. He stopped when he reached the vats of salad dressing, his stomach roiling with a mixture of nausea and panic.

"What can we do?" he asked.

"Queues," responded the professor. "Queues were practically designed for cafeterias."

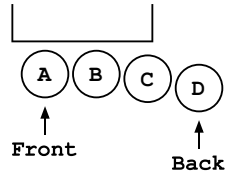
"Queues?" asked Frank.

"First-in, first-out data structures," explained Professor Heappens. "Like stacks, they also store things and have two operations. You can

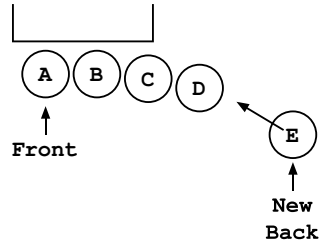
enqueue something by adding it to the back of the queue. Or you can *dequeue* something by taking it from the front. That way, you are always taking out the oldest item."

Frank tried to picture always taking the bottom plate from a stack. "But how?"

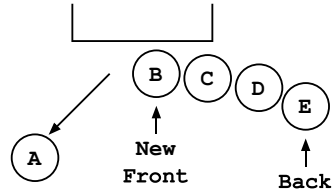
"That's just how the data structure works. Look at the sandwich line; it's a queue. Right now it has four people in it, and the person at the front has been waiting the longest."



Even as Professor Heappens said that, another person joined the line. "See, they enqueue at the back!" he noted.



They stood watching the line until the person at the front received her sandwich and departed.



"And dequeue at the front," said the professor happily. "What this cafeteria needs is more queues. Every cafeteria needs more queues."

Frank thought back to the mashed potato stack and realized the professor was right. How the data was stored could have a significant impact on how it was accessed. In cases like mashed potatoes, order mattered.

Despite the seemingly simple revelation, Frank struggled for days to integrate queues into the cafeteria. The plates and bowls were relatively easy. He would simply lift the old pile up and slide new plates underneath. Convincing the cooks to change how they ladled food proved more difficult. They thoroughly enjoyed ladling giant spoonfuls of potatoes, smiling as the large gobs smacked down into the bowl. Frank ultimately suggested a two-bowl method where the old potatoes were ladled onto the top of the new bowl. While it wasn't strictly a queue, it preserved all the fun of slopping mashed potatoes, and the old food didn't get buried at the bottom.

Unfortunately, disaster struck when he filled in for a sick baker. Not paying attention to the fact that the bread was baked in batches for a reason, Frank insisted that loading the oven last-in, first-out was unfair to the bread at the back. He devised a rotation scheme that, every 25 seconds, inserted a new loaf, rotated all the loaves in the oven, and removed the oldest loaf.

Frank's attempt at a baking queue might have worked if the oven had two doors, one at the front and one at the back. Unfortunately, the cafeteria used an older, single-door model, which made rotating the loaves in and out extremely difficult. While the constant churn ensured a more consistent cooking time for all loaves, Frank found himself unable to keep up with the schedule. Soon, dense smoke poured from the hearth as the loaves blackened.

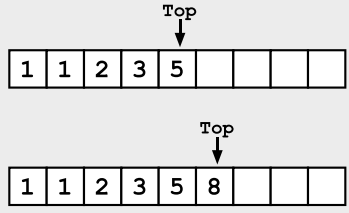
As the other cooks dashed to the fire with buckets of water, Frank stared numbly at the charred loaves. A sense of hopeless confusion crept in as he realized that queues might not be the solution to *every* cafeteria problem. He still had a lot to learn about data structures.

POLICE ALGORITHMS 101: STACKS AND QUEUES
Excerpt from Professor Drecker's Lecture

Stacks and queues are two simple structures for storing data. At first glance, both data structures resemble nothing more than lists of values. How these two data structures differ, though, is in how data is inserted or removed.

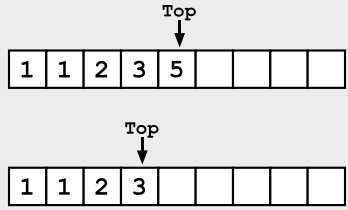
A stack is a last-in, first-out data structure that operates much like the pile of papers you'll find on every officer's desk. New elements are *pushed* onto the top of the stack, and elements are removed by being *popped* off the top of the stack. If five elements are pushed onto an empty stack in the order 1, 2, 3, 4, 5, they will be popped off in the reverse order, 5, 4, 3, 2, 1. Of course, as soon as your pile of papers is gone, your captain will just give you more paperwork.

You can implement stacks using an array and a single variable to track the index corresponding to the top of the stack. When you push a new element onto the stack, you add it to the next open slot in the array: index = top + 1. You also increment the top index accordingly.



When you pop an element off the stack, you can again use the top index to find the correct element. You can then remove this from the array and decrement the top index accordingly.

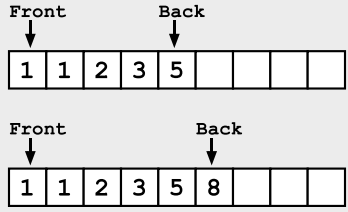
continued



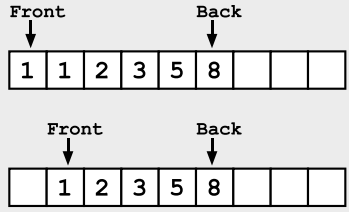
Of course, you must be careful when adding elements to an array of fixed size to avoid going past the end of the array.

A queue is a first-in, first-out data structure, much like a line of suspects waiting to be processed. New elements are *enqueued* at the back of the queue, and elements are removed by being *dequeued* from the front. If five elements are enqueued in an empty queue in the order 1, 2, 3, 4, 5, they will be dequeued in the same order, 1, 2, 3, 4, 5.

Queues can also be implemented with arrays. In this case, you need to track two indexes—the first and last element in the queue. When you enqueue a new element, you add it behind the current last element and increment the back index.



And when you dequeue an element, you remove the front element and increment the front index accordingly.



As you enqueue and dequeue elements in a fixed array, a block of empty space will build up at the front of the array. While you can design the queue to wrap, you must take care during both enqueuing and dequeuing to handle indexes being incremented past the end of the array.

— 13 —
Stacks and Queues for Search

Frank shook the image of burnt bread from his mind and returned to the present situation—trapped in a small cell filled with parchment about to burst into flame. The fire was still small, burning the loose sheets at the edge of the piles. But once the large stacks fully caught fire, the heat would be unbearable.

Socks crawled to the door and leaned against it. "Is it locked?" he asked.

Frank swallowed half a dozen snarky answers and simply nodded. "Can you open it?" he asked. "It's an old two-pin lock. It can't have that many combinations."

Socks shook his head. "There's no time. I know a spell to weaken the metal, though. It will ruin the door, but . . . I think that's okay given the circumstances and all."

He retrieved his staff and immediately set to work, mumbling incantations and running his hands over the bars. Spots of rust bloomed under his hands and crept over the metal. Less than a minute later, Socks stood back. The door looked thoroughly rusty, though still very much made of metal.

"The bars should be significantly weakened," he said. He stepped back and gave Frank an expectant look as if to say, "You can smash through the door anytime now."

Frank took a couple of steps back and eyed the door. "How weak?" he asked. "Are we talking toothpick weak or thick plank of wood weak?"

"Well . . . definitely weaker than normal metal," Socks answered. "I added a lot of rust. The bars are thick, but I think they should be pretty weak now."

Frank groaned. He took a deep breath and charged, lowering his shoulder and barrelling into the door. The impact jolted his entire body, but he broke through.

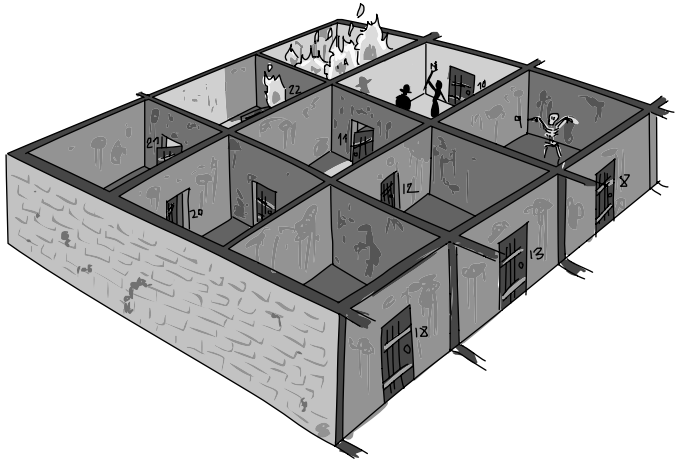
Frank lay sprawled on the floor while a cloudy mixture of rust particles and smoke swirled over him.

Socks hurried over to his side. "Are you okay?" He looked back at the door and broke into a wide smile. "It worked!" he said, beaming with pride. "Were they really weak? What did it feel like?"

"Like inch-thick pine," Frank said. "It hurt a lot."

The smile dimmed slightly. "Oh."

Frank pushed himself to his feet. His shoulder throbbed and he'd have a nasty bruise there tomorrow, but the temporary euphoria of escaping a flaming death easily offset the pain.



"Time to go," he said as he started through the next room.

"Do you remember how to get back?" Socks asked.

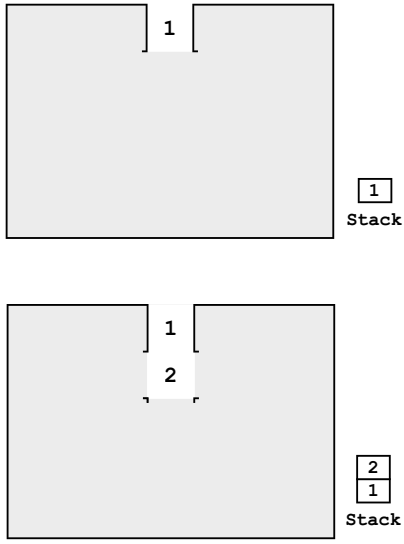
"Of course," Frank replied. "We used depth-first search to get here. We can just follow the stack back out."

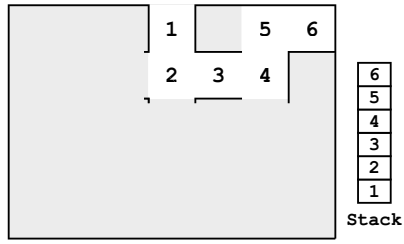
"Stack?" Socks asked as he started after Frank.

"Yeah," said Frank, still feeling the lingering thrill of their escape. "It's easy to think of searches in terms of the data structures they use. For example, breadth-first search uses a queue and depth-first search uses a stack." The explanation poured out of him like one of Notation's textbook answers.

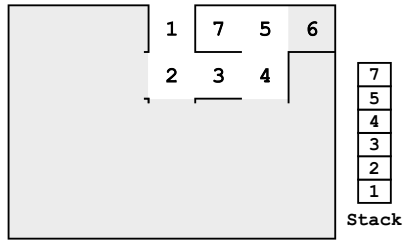
"Actually, there are a few different ways to keep track of your options during depth-first search. Some people prefer to use a stack to keep a list of *future* rooms to explore, similar to how you use a queue in breadth-first search. I prefer a different approach.

"You can use the stack to keep track of rooms along your *current* path. Every time you explore a new room, you push it onto a stack representing your current path.





"When you backtrack, you pop that room off the stack and return to the one before it. That way, you always know how to backtrack. I even numbered the rooms to make backtracking easier."

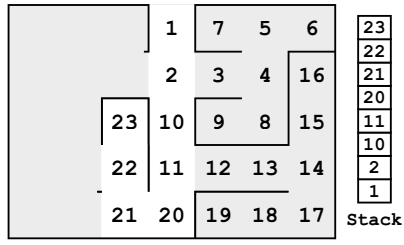


"I thought you always just backtracked to the last decision point," Socks said.

"You effectively do," said Frank. "But keeping the rooms in a stack makes it much easier to do that. You just backtrack and pop off the fully explored rooms until you get to one with a new path."

Socks looked impressed. "You wrote down the rooms we explored?"

"I kept track of the stack in my head and I numbered the rooms with chalk," answer Frank. "As I said, this isn't my first time doing depth-first search. We have to backtrack through seven rooms."



They hurried back through two dark rooms before Socks remembered the staff in his hand. He mumbled the fire incantation again and blue flame leapt from the tip.

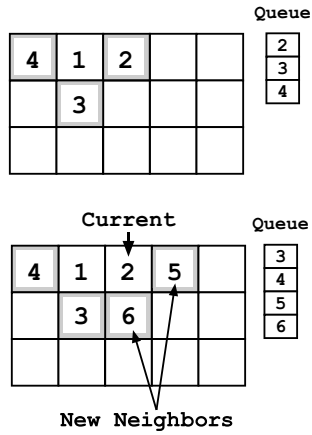
Frank eyed the staff warily. "Keep a tight hold of it this time," he advised.

After three more rooms, Socks suddenly asked, "What about queues?"

"What about them?" Frank asked.

"You said they were used for breadth-first search."

"They are," agreed Frank. "Your magic list was just a queue. In *breadth*-first search, the queue tracks the unexplored options. Instead of pushing the current state onto a stack, you add new neighbors to the back of a queue."



"And in *depth*-first search you can use your list, or stack, to track either the unexplored neighbors or the current path?" asked Socks, rather excitedly for someone fleeing an unknown attacker in an abandoned prison.

"Either approach will work if you're careful about the bookkeeping," agreed Frank.

"I had never thought of search in terms of stacks and queues before," Socks mused. "I wonder what other data structures I'm overlooking. I bet the Spell of Disentangled Ropes uses a few."

Frank ignored his ramblings and continued backtracking to the exit. They moved fast, prioritizing escape over further exploration. Simple logic told Frank that their attacker was long gone. No one had tried to stop them from escaping, and, with the evidence burning, there was nothing for the criminals to gain by waiting around.

Within a few minutes he located the final door, and they rushed outside. A thin trickle of smoke followed them out. By now the flames would have consumed the stacks of paper, destroying any leads.

POLICE ALGORITHMS 101: STACKS AND QUEUES

Excerpt from Professor Drecker's Lecture

The key to efficient algorithms is information. How we organize that information and the data structures we use can have a significant impact not only on the efficiency of the algorithm but also on how the algorithm actually functions. For a simple example of the importance of data structures, consider the breadth-first search and depth-first search from the previous lessons. While the algorithms are conceptually similar, whether we maintain our list of leads in either a stack or a queue significantly changes how the searches progress.

You want to be careful when choosing your data structures. Data structures should help enable the algorithm. Imagine what would happen if we stored a list of sorted numbers in a graph. Even if we maintain the sorted property, we can't perform an efficient binary search over the data because graphs limit how we access the data. Unlike arrays, graphs don't have indexes with which we can access the values. Instead, we are forced to perform a linear scan, moving from one node to the next via the graph's edges.

— 14 —
Let's Split Up: Parallelized Search

"What happened?" asked Officer Notation, who stood by the gate. Frank studied her as he tried to catch his breath. Was she concerned? Confused?

"We were attacked!" Socks blurted out. "We were trapped in a cell and everything was on fire! But I used a metal weakening spell so we could escape." He looked quite pleased with himself.

"Attacked?" asked Notation. "Who attacked you? Did you see them? What did they look like?"

"No," admitted Socks. "He snuck up behind me."

"Frank?" asked Notation, turning to Frank.

Frank shook his head. "All I saw was Socks flying at me."

"I bet he was big," offered Socks. "A giant thug. And he was stealthy. Maybe a trained assassin."

Frank rolled his eyes. "Sorry, kid. He was an amateur. Professional assassins don't lock people in cells and run away."

"But the fire," said Socks.

"Your staff started the fire," Frank reminded him. "You dropped it on the papers."

"Papers?" asked Notation. "Did you find the logs? Do you know what they were after?"

Frank and Socks looked at each other. Notation glanced from one to the other. Finally, Frank spoke, "The logs are gone. Our junior wizard here dropped his staff and *poof*—fire everywhere. Any clues are gone now."

Socks turned a deep shade of red and stared at the ground.

"Gone?" asked Notation. "Everything's gone? Are you sure?"

"Yeah," said Frank. He nodded toward the smoke trickling from the door.

"What about the person who attacked you?" asked Notation.

"I didn't get a look at him," said Frank. "I don't suppose *you* saw anything?" he asked. The question came out sharper than he'd intended, but after being attacked, getting trapped in a burning room, and fleeing a darkened prison, he didn't feel like pulling any punches.

"No," she said calmly. "There was nothing around the front."

"No tracks near any of the doors?" asked Frank. "Anything that could give us a clue about our attacker?"

Notation shook her head. "Nothing," she said. "It looked as though no one had been around the other side in months."

Frank nodded, but didn't speak. Something felt wrong. Either the attacker had cleverly slipped by Notation through the same gate they had used, or she wasn't telling them something. How long had she been away from the gate? And why had she stayed outside? Frank decided not to press the issue. "All right. Let's head back to the boat."

"Now what?" asked Socks as they made their way toward the water.

"Time to backtrack," said Frank. "No more clues here."

"Backtrack to where?"

"The open clues," Frank answered. "We investigate the leads we still have left." He paused for a moment, weighing his options. "I think it's time we parallelize the search."

"Really?" asked Notation.

"Parallelize?" asked Socks.

"It means we split up and explore different parts of the search space," answered Notation. "Parallel algorithms divide up work and do that work in parallel—at the same time, that is. For example, the work might be distributed over different people. In this case, we can divide the leads into three sets. Then you, Frank, and I can each take a set of leads. We can investigate different leads at the same time, allowing us to work almost three times faster."

"But," objected Socks, "I'm not an officer or a private investigator. I don't know what to do. Shouldn't I stay with one of you?"

"No," said Frank. "I'm still not sure what's going on, but I have a feeling we're working with limited time. Whoever we're after knows we're on the case now and knows we've tracked them this far. If they're smart, they'll start destroying the rest of the evidence."

"It'll be late by the time we get back to Usb," noted Socks.

"We can split up tonight and meet at my office tomorrow morning," said Frank. "That should give us enough time to follow leads and possibly grab some sleep."

"Okay," agreed Notation. "How do we divide up the work?"

Frank knew the key to an efficient parallel algorithm was making sure that the benefit of using multiple workers was worth the cost of dividing up the work. Parallelizing work involves a certain amount of overhead. The problem needs to be broken into pieces. Each worker has to get a task, get ready to do it, and then actually do it. And then at the end, the work has to be recombined. Parallelizing a simple task can sometimes be more costly than just solving it outright. However, when the problem gets large enough, parallelization can greatly accelerate an algorithm.

"Easy," said Frank. "Socks, I need you to talk to your wizard friends. Ask them if they know about a group called the League of something. The thugs on the boat said they were working for a league before Rebecca Vinettee interrupted them. Judging by previous cases, the name will be something evil like the League of

Power-Hungry Maniacs or the League of Darkness. Evil leagues tend not to be subtle with their names. Find out everything you can about this group."

"That's not much to go on," Socks complained.

"Notation," continued Frank. "I need you to pull all police transfer records for the last six months." While he didn't like the idea of leaving this lead to Notation, she was the only one who could get the records easily. If he tried to collect them himself, he would be met with suspicious looks and a small barricade of paperwork. The capital police department used paperwork with the same determination and effectiveness as roadblocks.

"Transfers?" asked Notation, clearly surprised. "Why?"

"Call it a hunch," Frank lied. "We'll meet at my office tomorrow morning and combine our information."

"What about you?" asked Notation. Her voice now carried a note of irritation. She obviously knew Frank wasn't telling her the whole story.

Frank gave her an innocent smile. "I have to go shopping."

As the *TCP Flyer* slowly cruised back to Usb, Frank found an out-of-the-way corner on the deck and sat down to think. This was the part of the investigation he hated most, when promising leads started drying up or, in this case, burning up. The loss of a crucial clue always filled Frank with a sense of dread—like he was running one step behind. Frank forced the doubts from his mind and refocused on the clues he did have. The voyage to Usb would give him time to comb through what he had seen and find connections he had missed.

He closed his eyes and took a deep breath.

"Oh. Sorry. Are you sleeping?" asked Socks.

"No. Thinking," Frank said, and congratulated himself for not yelling. After all, the boy had saved his life.

When Socks didn't say anything more, Frank prompted, "What do you want, Socks?"

"Um . . . I was curious about the search," Socks responded.

"How so?" said Frank.

To Frank's dismay, Socks walked over and sat next to him.

"Do you think we'll find the criminals?" asked Socks.

Frank shrugged. "We've still got some good leads," he offered.

"But do you think we'll be in time?" asked Socks.

Alarm bells went off in Frank's head. He swiveled and stared hard at Socks. "Time for what?"

Socks nearly fell backward. His eyes darted around, as though searching for an appropriate answer. "Whatever they're planning?" he finally stammered.

Frank didn't buy it. "What else do you know?" he asked.

"Nothing," replied Socks. "At least, nothing concrete. It's just speculation. Not mine—my mentor Gretchen's. She has good insight into these types of things, though."

"Which is?"

"I really shouldn't say anything. It's just speculation."

"*Which is?*" Frank growled.

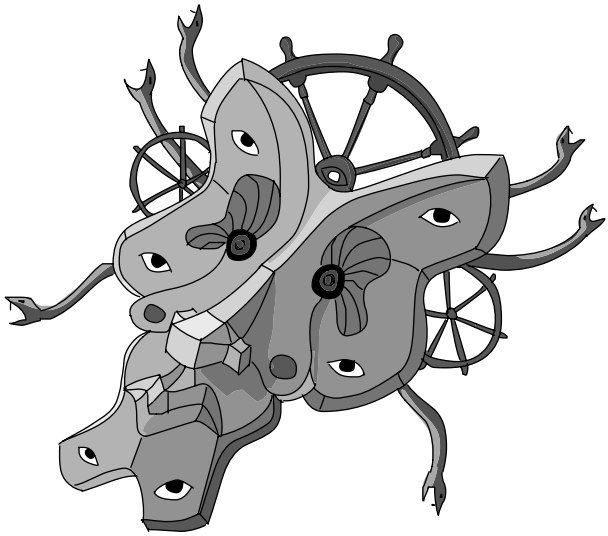
"She thinks that whoever is behind this is going to attack the castle in a few days."

Frank leapt to his feet. "Why didn't you mention this sooner?" he shouted.

"It's just speculation," repeated Socks.

"Unless it is a complete guess, she must have some reason," said Frank. "Is it a guess?"

"No. Not entirely," said Socks. "It's based on the stolen mask. Magical artifacts are most effective during the full moon, which is in two days."



"What exactly does this mask do?" asked Frank, starting to pace anxiously.

Socks hesitated for a moment. "It's an incredibly powerful artifact," he started. Upon seeing the angry look in Frank's eyes, he sped up. "It's officially called the Mask of Combinatorial Looks. It was lost hundreds of years ago during the Great Slug War. Everyone thought it was destroyed until Princess Ann recovered it during one of her quests. She found it—"

"What does it *do*?" prompted Frank.

"It allows the wearer to look like anyone else. The scholars believe that it uses a massively parallel search. Each feature runs its own search to find the best match. The nose will transform into a perfect match of the target's nose. The eyes will transform into—"

"A perfect disguise," offered Frank.

"Yes," said Socks.

Frank cursed. "And the castle? Why does Gretchen think they'll attack the castle?"

"She didn't say," admitted Socks. "Maybe that part *was* a guess," he added without any real conviction.

Frank didn't believe it either.

"I'm sorry I didn't mention it earlier," offered Socks. "Since there isn't any hard proof . . ." He trailed off, looking miserable.

"What else aren't you telling us?" asked Frank, staring down at Socks.

Socks thought about the question for a long while before answering. "I think that's it."

"Everything?"

"Everything I know," Socks qualified.

Frank took a deep breath and looked up at the sails, wishing they were fuller. The wind had died down within the last hour, and the *TCP Flyer* seemed to be barely inching toward their destination.

He ran the timeline of the next few days through his head and wondered if they had enough time. Even with three of them searching in parallel, there was no guarantee they would cover enough ground. Worse, they couldn't even begin the parallel search until the *TCP Flyer* docked. Until then, they were all stuck on the boat.

POLICE ALGORITHMS 101: PARALLEL ALGORITHMS

Excerpt from Professor Drecker's Lecture

A parallel algorithm breaks up a problem into multiple pieces, performs the computation on those pieces at (approximately) the same time, and then combines the results when all pieces are finished. It divides up the work among different workers, allowing them to complete the task faster than a single worker could. Consider our favorite example: searching an abandoned building for a suspect. The more officers you have, the more rooms you can check at the same time and the faster you can find the suspect.

continued

If you have 30 rooms and 30 officers, they can kick in all the doors at once.

The key to an efficient parallel algorithm is to efficiently divide the work into independent units and then recombine it. Some problems are trivial to parallelize. For example, if you are searching a large stack of scrolls for a particular clue, you can easily divide the work by giving each worker a subset of the scrolls.

However, other algorithms are much more difficult or even impossible to parallelize. Even if you have 100 officers, you can't question a suspect any faster. It's an inherently serial problem. You need to base your next question on the suspect's previous answers. And, perhaps more importantly, a suspect can answer only one question at a time. I've seen eight officers shouting questions at the same time. The interrogation doesn't go any faster.

Another aspect to consider when parallelizing algorithms is whether the efficiency is even worth the overhead. A parallel algorithm requires additional setup time to divide the work, as well as completion time to merge the results back together. Individual tasks have to be assigned to different workers, often requiring some amount of communication. Consider the task of searching an unsorted array with only three values. By the time the setup is complete, a single person could have likely scanned through the array many times over.

— 15 —
Iterative Deepening Can Save Your Life

I know that look," said Mavis. Frank looked up at the *TCP Flyer*'s captain in annoyance. He preferred to brood quietly, and this was the second interruption in 10 minutes.

"What look?" he growled.

"That look," she said, waving in Frank's general direction. "You're questioning your search and wondering if you spent too much time on dead ends."

"Why would I be doing that?" Frank asked.

"I heard what the kid said," Mavis explained. "You're suddenly on a tight timeline, and we have at least another hour before we get back to Usb."

Frank nodded. "If this piece of junk—"

"Hey, now. Just because you're questioning your search doesn't give you a reason to insult my ship."

"Yeah. I suppose," Frank mumbled by way of apology.

He had been running through the leads in his mind, wondering if one of them would have provided quicker answers. He knew the log entries were good leads—as good as he could hope to find in a case like this. But they'd been time-consuming. He'd spent almost a full day traveling between ports on the *TCP Flyer*.

With a grunt, Mavis lowered herself and sat next to Frank. "Iterative deepening?"

Frank shrugged. The thought had occurred to him. Iterative deepening was a cross between a pure depth-first search and a breadth-first search. The algorithm searched in rounds, each round being a depth-first search that was limited to a given path length.

"Never was a fan," Frank admitted. He'd never been able to stomach repeating parts of the search over and over each iteration. So much of the work seemed to be wasted.

Mavis laughed. "You haven't faced enough dead ends then."

Frank raised an eyebrow. "You're talking to a private investigator. I run into more dead ends than correct paths."

"Ever lose a criminal because of one?" asked Mavis.

"A few times," Frank admitted.

"Then you should appreciate iterative deepening," said Mavis. "When I first saw it in action, I was annoyed by the restarts too. But it's saved my life more than once."

"Restarting a search over and over saved your life?" asked Frank.

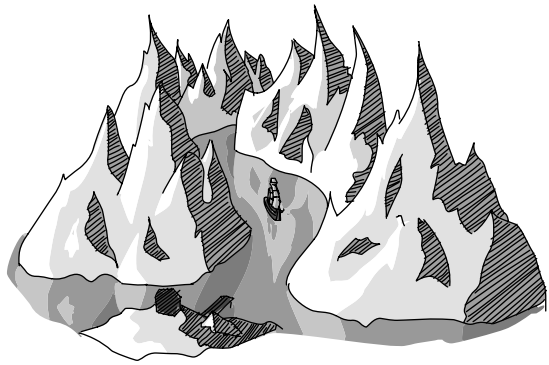
"Limiting how far I could explore along the wrong path saved my life," corrected Mavis.

"When did iterative deepening save your life?" asked Frank, unable to keep the skepticism from his voice.

Mavis stared out over the ocean. "Well . . . the first time was when I was just a kid. I was an apprentice on a cargo vessel called the *Void Star*. It was an amazing ship; it could carry anything. Anyway, we were lost in the middle of the Razor Ridges—a dense series of volcanic peaks that effectively form a giant maze—and we were running out of important provisions."

"Water?" asked Frank.

"No," answered Mavis. "We had at least two weeks' worth of food and water. We were low on coffee, and that was bad news for the ship's officers. After a single day without coffee, the first mate would get twitchy and sing depressing sea shanties."

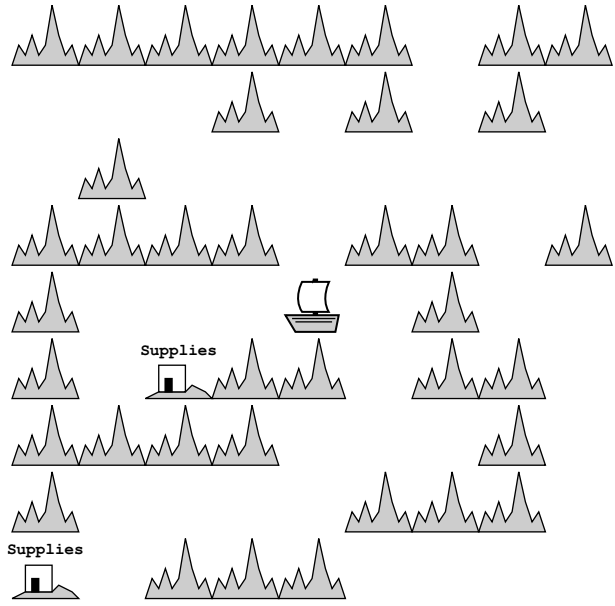


"That doesn't sound too bad."

"Without coffee, the man's singing attracted every vicious bird in an eight-mile radius."

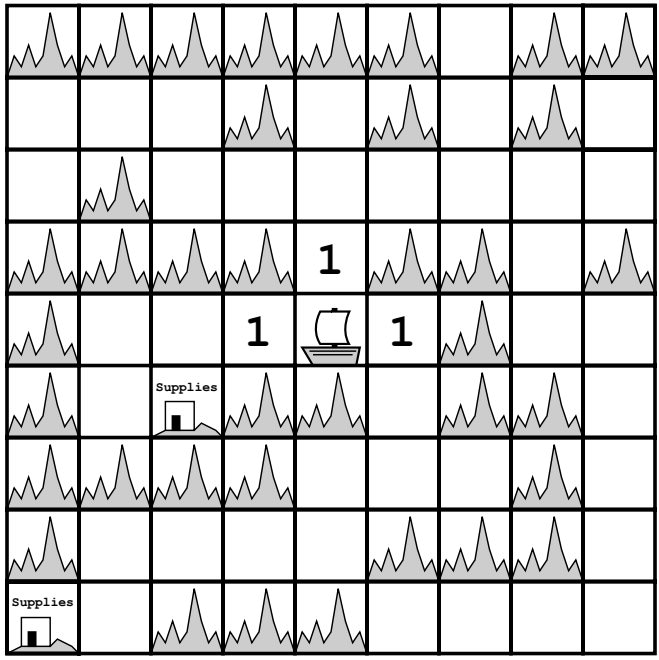
Frank winced at the thought.

"Anyway," continued Mavis, "coffee was vital for the ship. The captain estimated we had less than two days to find an island with a supply station. She knew there had to be one close by, but didn't know exactly where. You see, we'd lost the map during an impromptu paper airplane contest. And with the dense fog throughout the ridges, we wouldn't see the station until we were right on top of it."

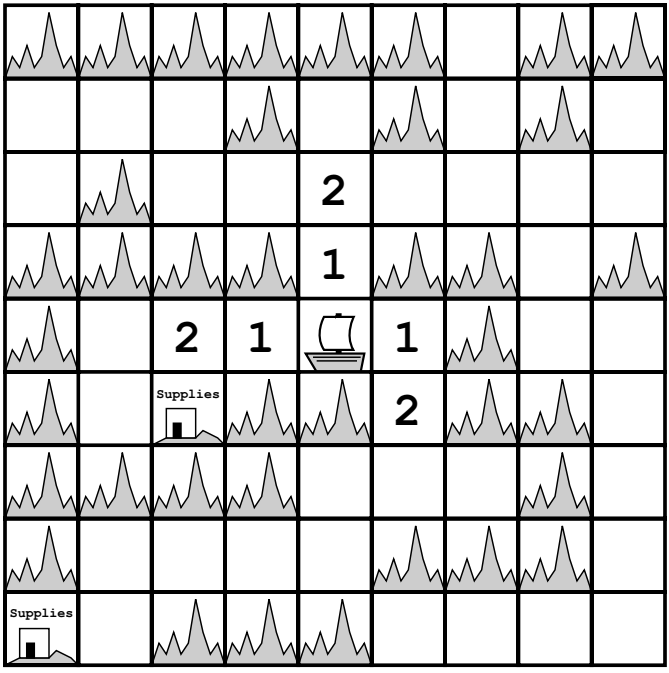


"We started off searching for an island with coffee. I was still green in those days and hadn't heard of iterative deepening, so I boldly suggested a depth-first search. The captain just laughed and told me she'd never trust a depth-first search in the Razor Ridges—too many long dead ends.

"Well. She gridded off the sea into one-mile-square chunks. One mile was about as far as you could see through the fog, so we would need to be in the same grid square as the supply station to see it. Then we set about exploring with iterative deepening. We used a depth-first search, but limited it to a single one-cell step. We used a classic north, east, south, west ordering, backtracking to the starting location each time. We didn't find anything in this first step, but at least we were efficient about it. Within a few hours, we had eliminated all neighboring grid squares.



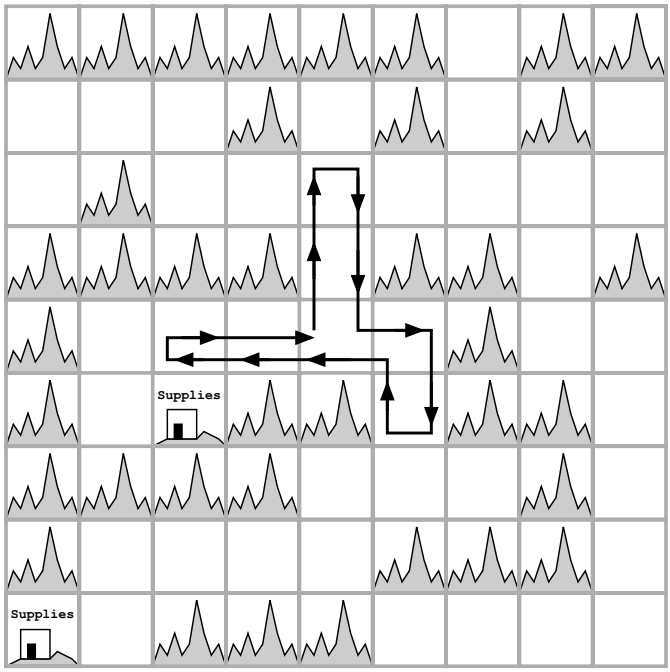
Mavis shook her head. “No sign of the supply station at all. So we started over, doing another depth-first search from the original starting point. This time we explored two steps out and covered a lot more area. We ended up reexploring the neighboring squares in the process. Still no sign of a supply station, but we were able to eliminate all squares within two steps pretty quickly.”



“Why not just use breadth-first search?” asked Frank. “That’s what you were effectively doing anyway. Your search explored outward, farther and farther from the starting point.”

Mavis nodded. “Breadth-first search and iterative deepening have a lot in common. But you’re forgetting one key point. *We had lost our map.* It’s really difficult to track your unexplored states in breadth-first search when you don’t have a map. How do you remember your

frontier? Iterative deepening allowed us to explore outward without having to explicitly remember all the unexplored states. We just followed a depth-limited path."

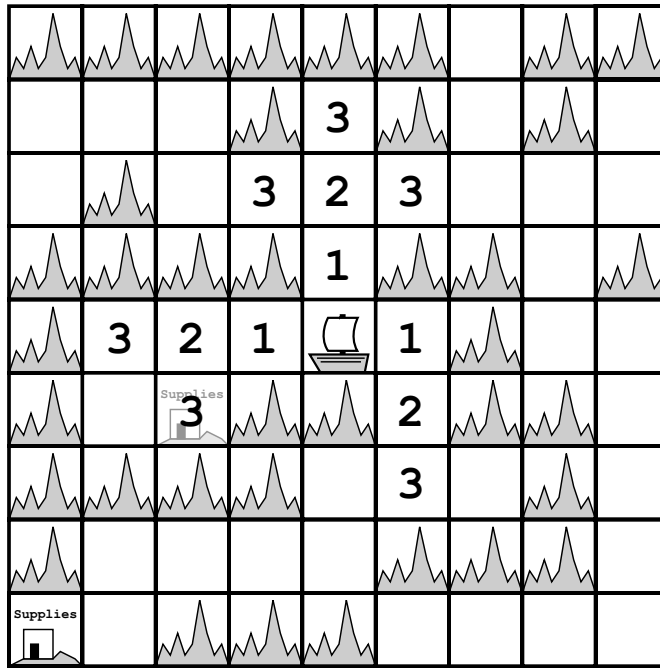


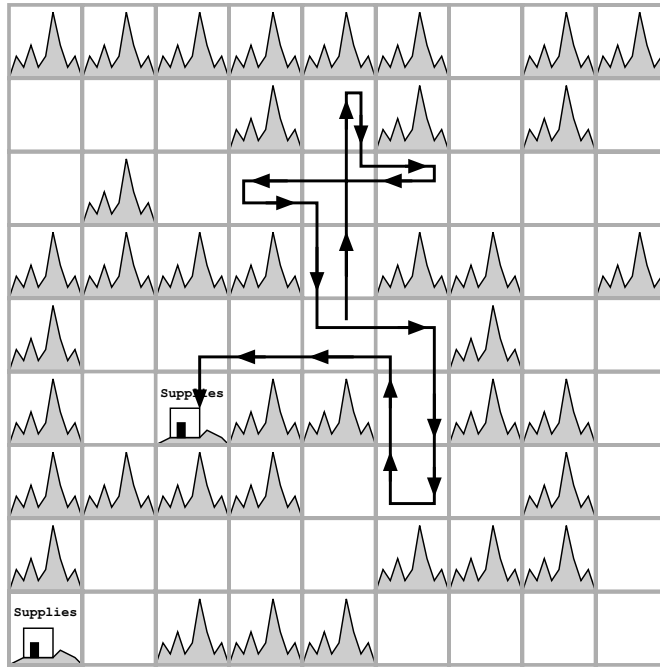
"I guess so," Frank agreed.

"Anyway, we were starting to run low on coffee at that point," Mavis continued. "A group of volunteers, including the captain herself, switched to decaf. But we all knew that would only buy us a little time. We pushed on. We restarted the depth-first search again, this time allowing ourselves to venture out farther."

"Did you find it with a search of length three?" asked Frank.

"Luckily, we did," replied Mavis. "On that iteration we checked everything one, two, and three steps away. By that time, the quartermaster, who had absolutely no use for decaf, had resorted to reusing the same grounds for a 10th time, but the first mate was already singing 'Sea Slugs on Deck.' Fortunately, that was one of his more upbeat tunes."





Frank thought about it for a moment. "What if you had skipped the repeated work? What if you had just used depth-first search?"

"We would have gone down a long dead end and run out of coffee," she replied. "Didn't I start by telling you it saved my life?"

"Fair enough. But that's a matter of luck. The nearest supply station could have been down a depth-first search of length five."

"Ha! You know better than that, Frank. You can always find lucky or unlucky problems. Iterative deepening can help you hedge against really unlucky cases. It bounds how far you away you can go on any iteration."

"Other algorithms do that too," he countered.

Mavis scowled. "I didn't say iterative deepening was the *only* algorithm that could have saved us. I said it was the one we used. And I have used it ever since.

"Once I even used it to track down an angry shoal of squid before they inked the capital's harbor. Oh, it would have been a grand mess. Some days I wonder if I should have just let them do it. The king's reaction would have been priceless."

Frank thought for a long while, wondering if iterative deepening could have saved him time here. By cutting off the search sooner, he could have backtracked and followed up on the threads or the mysterious league. But then he wouldn't have been following the highest-priority lead.

He shook his head. "I'll stick with my usual searches," he said finally.

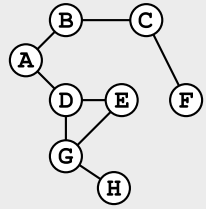
Mavis nodded solemnly and looked out over the ocean. "Fair enough. But be careful, Frank. You don't have much time and long dead ends can be costly. With any algorithm, you should at least think about how to protect yourself from running into the worst-case problems."

POLICE ALGORITHMS 101: ITERATIVE DEEPENING

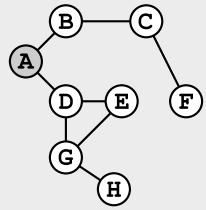
Excerpt from Professor Drecker's Lecture

Iterative deepening is a modification of depth-first search that repeatedly performs limited depth-first searches. During iteration (or round) k of iterative deepening, the algorithm performs a depth-limited search with *max-depth* = k.

Consider again the example of searching for a suspect starting from city A.

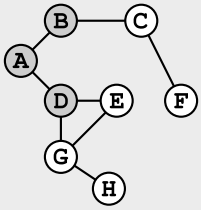


We start with a depth-first search but cut it off after the first node, A. This corresponds to limiting ourselves to searching just the scene of the crime.

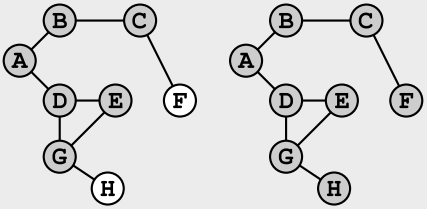


The next iteration restarts the depth-first search but allows it to explore one city away. We cover the close cities, visiting A, B, and D.

continued



As the search progresses, we have to go farther and farther from the scene of the crime. We end up searching the nearby cities multiple times on different iterations of our search. In fact, we search A four times and B three times.



While the repeated work increases computational cost, iterative deepening has advantages. It combines the lower memory requirements of depth-first search with the abilities of breadth-first search to find short paths and avoid getting stuck on some worst-case problems.