Chapter 1

Prologue

Some citizens of Königsberg
Were walking on the strand
Beside the river Pregel
With its seven bridges spanned.

“O Euler, come and walk with us,”
Those burghers did beseech.
“We’ll roam the seven bridges o’er,
And pass but once by each.”

“It can’t be done,” thus Euler cried.
“Here comes the Q.E.D.
Your islands are but vertices
And four have odd degree.”

William T. Tutte

1.1 Crossing Bridges

We begin our journey into the nature of computation with a walk through 18th-century Königsberg (now Kaliningrad). As you can see from Figure 1.1, the town of Königsberg straddles the river Pregel with seven bridges, which connect the two banks of the river with two islands. A popular puzzle of the time asked if one could walk through the city in a way that crosses each bridge exactly once. We do not know how hard the burghers of Königsberg tried to solve this puzzle on their Sunday afternoon walks, but we do know that they never succeeded.

It was Leonhard Euler who solved this puzzle in 1736. As a mathematician, Euler preferred to address the problem by pure thought rather than by experiment. He recognized that the problem depends only
on the set of connections between the riverbanks and islands—a *graph* in modern terms. The graph 1.1
corresponding to Königsberg has four vertices representing the two riverbanks and the two islands, and seven edges for the bridges, as shown in Figure 1.2. Today, we say that a walk through a graph that crosses

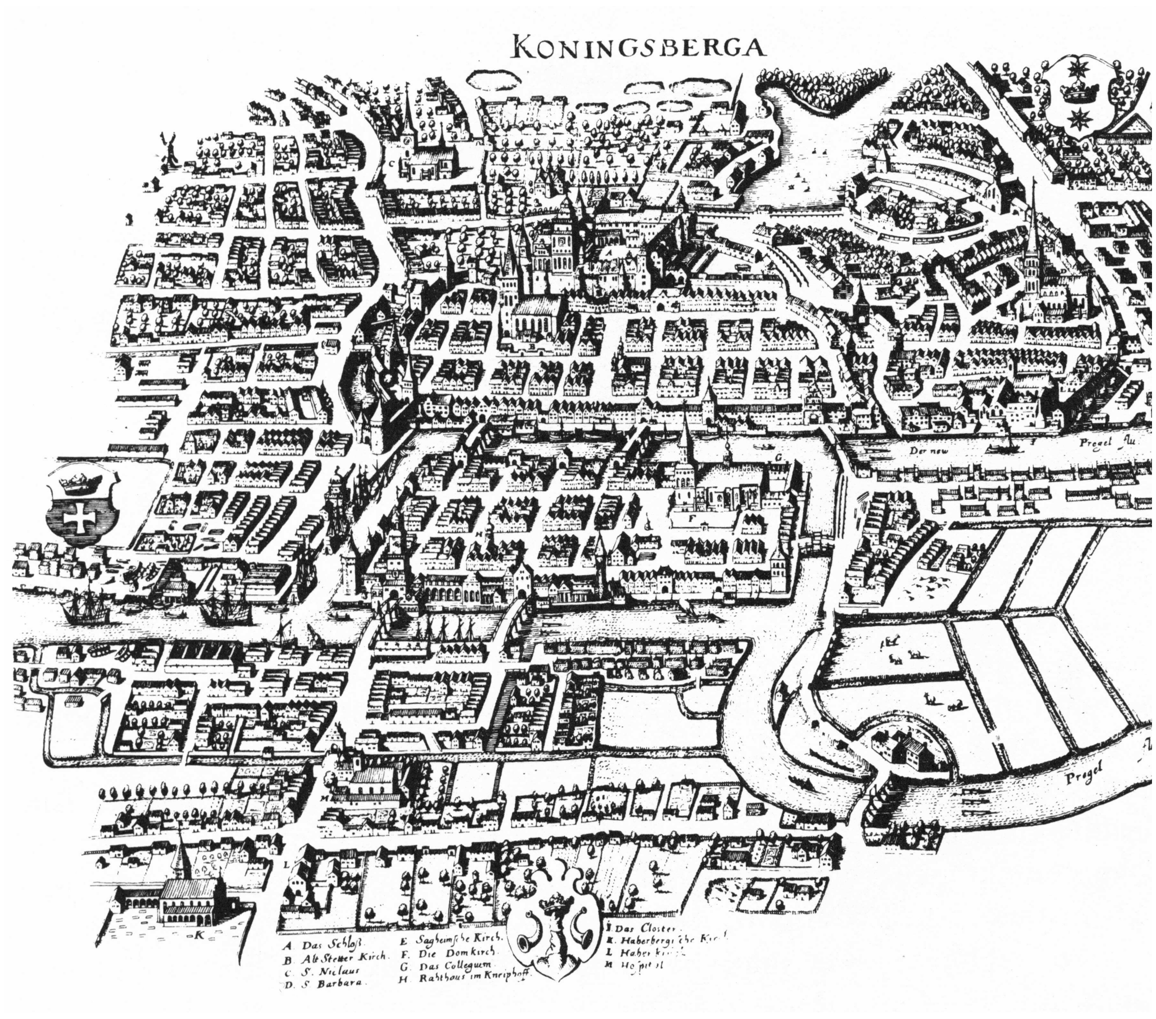


FIGURE 1.1: Königsberg in the 17th century.

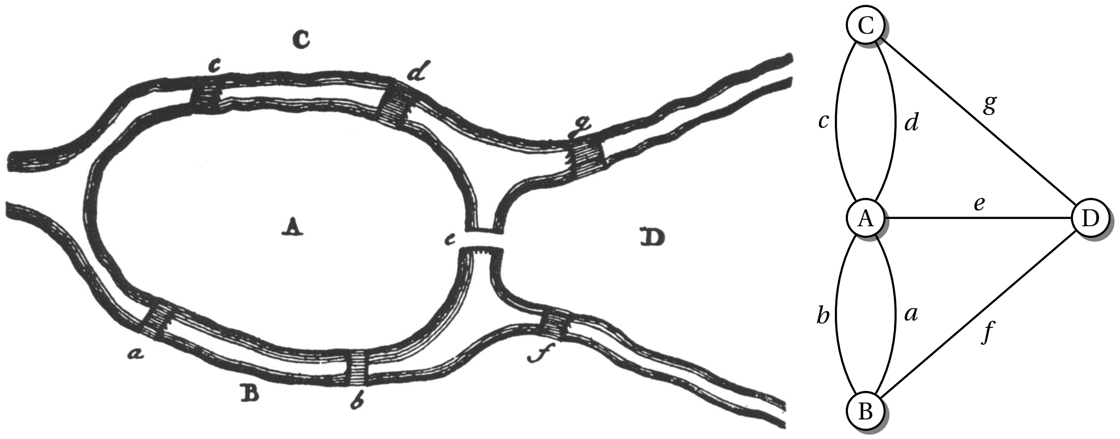


FIGURE 1.2: The seven bridges of Königsberg. Left, as drawn in Euler's 1736 paper, and right, as represented as a graph in which each riverbank or island is a vertex and each bridge an edge.

each edge once is an *Eulerian path*, or an *Eulerian cycle* if it returns to its starting point. We say that a graph is Eulerian if it possesses an Eulerian cycle.

Now that we have reduced the problem to a graph that we can doodle on a sheet of paper, it is easy to explore various walks by trial and error. Euler realized, though, that trying all possible walks this way would take some time. As he noted in his paper (translated from the Latin):

> As far as the problem of the seven bridges of Königsberg is concerned, it can be solved by making an exhaustive list of possible routes, and then finding whether or not any route satisfies the conditions of the problem. Because of the number of possibilities, this method of solutions would be too difficult and laborious, and in other problems with more bridges, it would be impossible.

Let's be more quantitative about this. Assume for simplicity that each time we arrive on an island or a riverbank there are two different ways we could leave. Then if there are n bridges to cross, a rough estimate for the number of possible walks would be 2^n. In the Königsberg puzzle we have $n = 7$, and while 2^7 or 128 routes would take quite a while to generate and check by hand, a modern computer could do so in the blink of an eye.

But Euler's remark is not just about the bridges of Königsberg. It is about the entire *family of problems of this kind*, and how their difficulty grows, or *scales*, as a function of the number of bridges. If we consider the bridges of Venice instead, where $n = 420$, even the fastest computer imaginable would take longer than the age of the universe to do an exhaustive search. Thus, if searching through the space of all possible solutions were the only way to solve these problems, even moderately large cities would be beyond our computational powers.

Euler had a clever insight which allows us to avoid this search completely. He noticed that in order to cross each edge once, any time we arrive at a vertex along one edge, we have to depart on a different edge. Thus the edges of each vertex must come in pairs, with a "departure" edge for each "arrival" edge. It follows that the *degree* of each vertex—that is, the number of edges that touch it—must be even. This

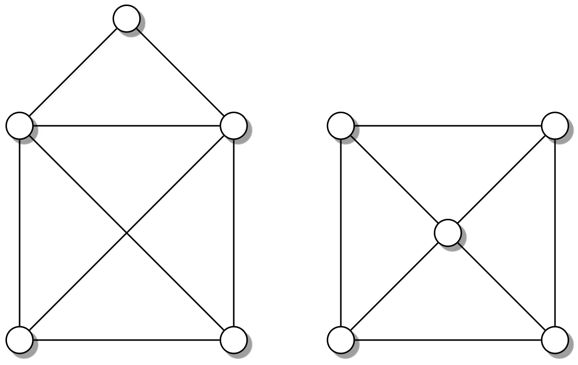


FIGURE 1.3: A classic children's puzzle. Can you draw these graphs without lifting the pen from the paper, or drawing the same edge twice? Equivalently, do they have Eulerian paths?

holds for all vertices except the vertices where the path starts and ends, which must have odd degree unless they coincide and the path is a cycle.

This argument shows that a necessary condition for a graph to be Eulerian is for all its vertices to have even degree. Euler claimed that this condition is also sufficient, and stated the following theorem:

Theorem 1.1 *A connected graph contains an Eulerian cycle if and only if every vertex has even degree. If exactly two vertices have odd degree, it contains an Eulerian path but not an Eulerian cycle.*

This theorem allows us to solve the bridges of Königsberg very quickly. As the poem at the head of this chapter points out, all four vertices have odd degree, so there is no Eulerian path through the old town of Königsberg.

Beyond solving this one puzzle, Euler's insight makes an enormous difference in how the complexity of this problem scales. An exhaustive search in a city with n bridges takes an amount of time that grows *exponentially* with n. But we can check that every vertex has even degree in an amount of time proportional to the number of vertices, assuming that we are given the map of the city in some convenient format. Thus Euler's method lets us solve this problem in *linear* time, rather than the exponential time of a brute-force search. Now the bridges of Venice, and even larger cities, are easily within our reach.

Exercise 1.1 *Which of the graphs in Figure 1.3 have Eulerian paths?*

In addition to the tremendous speedup from exponential to linear time, Euler's insight transforms this problem in another way. What does it take to *prove* the existence, or nonexistence, of an Eulerian path? If one exists, we can easily prove this fact simply by exhibiting it. But if no path exists, how can we prove that? How can we convince the people of Königsberg that their efforts are futile?

Imagine what would have happened if Euler had used the brute-force approach, presenting the burghers with a long list of all possible paths and pointing out that none of them work. Angry at Euler for spoiling their favorite Sunday afternoon pastime, they would have been understandably skeptical. Many would have refused to go through the tedious process of checking the entire list, and would have held out

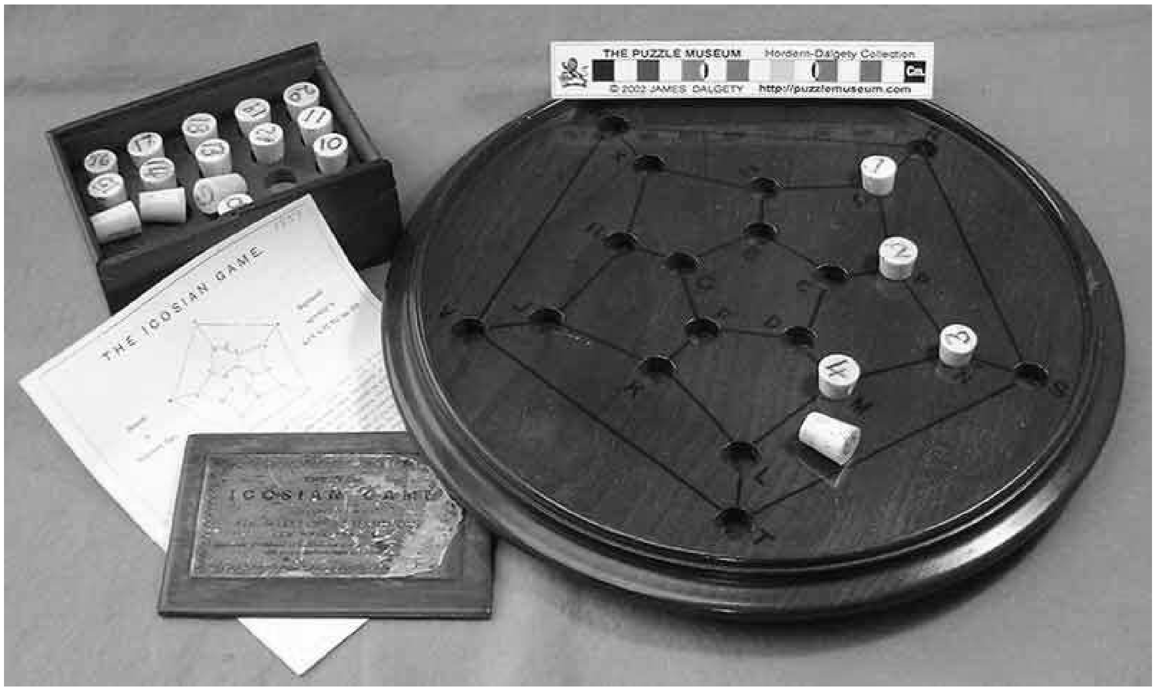

THE PUZZLE MUSEUM Hordern-Dalgety Collection
© 2002 JAMES DALGETY http://puzzlemuseum.com
THE ICOSIAN GAME.
ICOSIAN GAME

FIGURE 1.4: Hamilton's Icosian game.

the hope that Euler had missed some possibility. Moreover, such an ungainly proof is rather unsatisfying, even if it is logically airtight. It offers no sense of *why* no path exists. 1.2

In contrast, even the most determined skeptic can follow the argument of Euler's theorem. This allows Euler to present a proof that is simple, compact, and irresistible: he simply needs to exhibit three vertices with odd degree. Thus, by showing that the existence of a path is equivalent to a much simpler property, Euler radically changed the *logical structure* of the problem, and the type of proof or disproof it requires.

1.2 Intractable Itineraries

The next step in our journey brings us to 19th-century Ireland and the Astronomer Royal, Sir William Rowan Hamilton, known to every physicist through his contributions to classical mechanics. In 1859, Hamilton put a new puzzle on the market, called the "Icosian game," shown in Figure 1.4. The game was a commercial failure, but it led to one of the iconic problems in computer science today.

The object of the game is to walk around the edges of a dodecahedron while visiting each vertex once and only once. Actually, it was a two-player game in which one player chooses the first five vertices, and the other tries to complete the path—but for now, let's just think about the solitaire version. While such walks had been considered in other contexts before, we call them *Hamiltonian* paths or cycles today, and 1.3

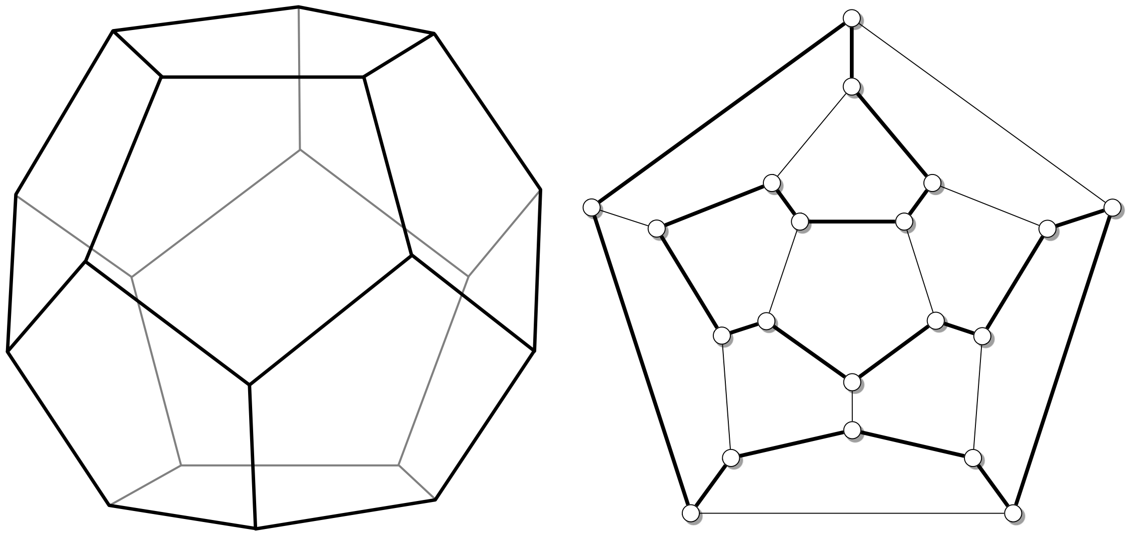


FIGURE 1.5: Left, the dodecahedron; right, a flattened version of the graph formed by its edges. One Hamiltonian cycle, which visits each vertex once and returns to its starting point, is shown in bold.

we say that a graph is Hamiltonian if it possesses a Hamiltonian cycle. One such cycle for the dodecahedron is shown in Figure 1.5.

At first Hamilton's puzzle seems very similar to the bridges of Königsberg. Eulerian paths cross each edge once, and Hamiltonian paths visit each vertex once. Surely these problems are not very different? However, while Euler's theorem allows us to avoid a laborious search for Eulerian paths or cycles, we have no such insight into Hamiltonian ones. As far as we know, there is no simple property—analogous to having vertices of even degree—to which *Hamiltonianness* is equivalent.

As a consequence, we know of no way of avoiding, essentially, an exhaustive search for Hamiltonian paths. We can visualize this search as a tree as shown in Figure 1.6. Each node of the tree corresponds to a partial path, and branches into child nodes corresponding to the various ways we can extend the path. In general, the number of nodes in this search tree grows exponentially with the number of vertices of the underlying graph, so traversing the entire tree—either finding a leaf with a complete path, or learning that every possible path gets stuck—takes exponential time.

To phrase this computationally, we believe that there is no program, or *algorithm*, that tells whether a graph with n vertices is Hamiltonian or not in an amount of time proportional to n, or n^2, or any polynomial function of n. We believe, instead, that the best possible algorithm takes exponential time, 2^{cn} for some constant $c > 0$. Note that this is not a belief about how fast we can make our computers. Rather, it is a belief that finding Hamiltonian paths is *fundamentally harder* than finding Eulerian ones. It says that these two problems differ in a deep and qualitative way.

While finding a Hamiltonian path seems to be hard, *checking* whether a given path is Hamiltonian is easy. Simply follow the path vertex by vertex, and check that it visits each vertex once. So if a computationally powerful friend claims that a graph has a Hamiltonian path, you can challenge him or her to

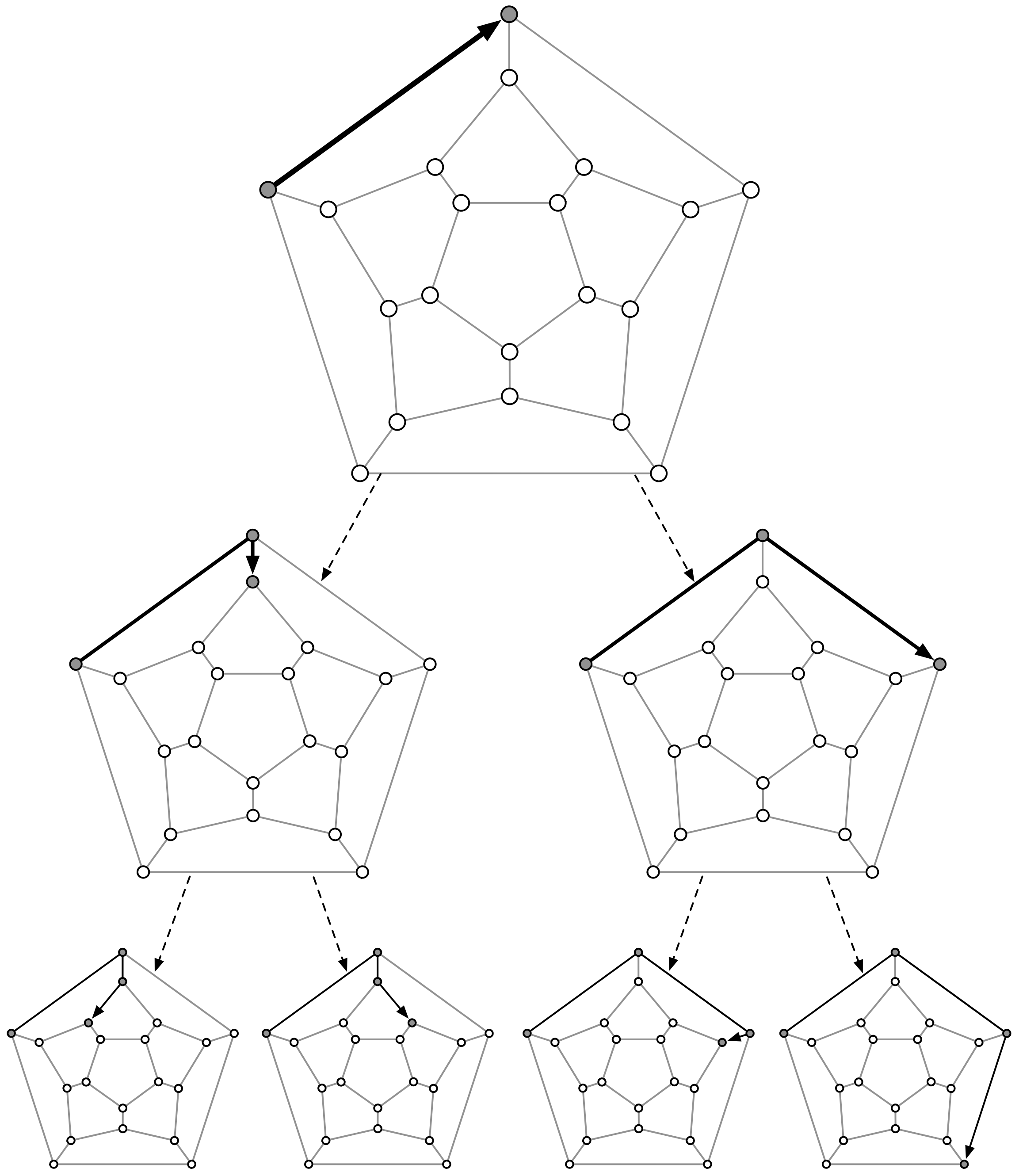


FIGURE 1.6: The first two levels of the search tree for a Hamiltonian path.

prove that fact by showing it to you, and you can then quickly confirm or refute their claim. Problems like these are rather like finding a needle in a haystack: if I show you a needle you can confirm that it is one, but it's hard to find a needle—at least without a magnetic insight like Euler's.

On the other hand, if I claim that a haystack has no needles in it, the only way to prove this is to sift carefully through all the hay. Similarly, we know of no way to prove that no Hamiltonian cycle exists without a massive search. Unlike Eulerian cycles, where there is a simple proof in either case, the logical structure of Hamiltonianness seems to be fundamentally asymmetric—proving the nonexistence of a Hamiltonian cycle seems to be much harder than proving its existence.

Of course one might think that there is a more efficient method for determining whether a graph is Hamiltonian, and that we have simply not been clever enough to find it. But as we will see in this book, there are very good reasons to believe that *no such method exists*. Even more amazingly, if we are wrong about this—if Hamilton's problem can be solved in time that only grows polynomially—then so can thousands of other problems, all of which are currently believed to be exponentially hard. These problems range from such classic search and optimization problems as the Traveling Salesman problem, to the problem of finding short proofs of the grand unsolved questions in mathematics. In a very real sense, the hardness of Hamilton's problem is related to our deepest beliefs about mathematical and scientific creativity. Actually *proving* that it is hard remains one of the holy grails of theoretical computer science.

1.3 Playing Chess With God

It's a sort of Chess that has nothing to do with Chess, a Chess that we could never have imagined without computers. The Stiller moves are awesome, almost scary, because you know they are the truth, God's Algorithm—it's like being revealed the Meaning of Life, but you don't understand one word.

Tim Krabbé

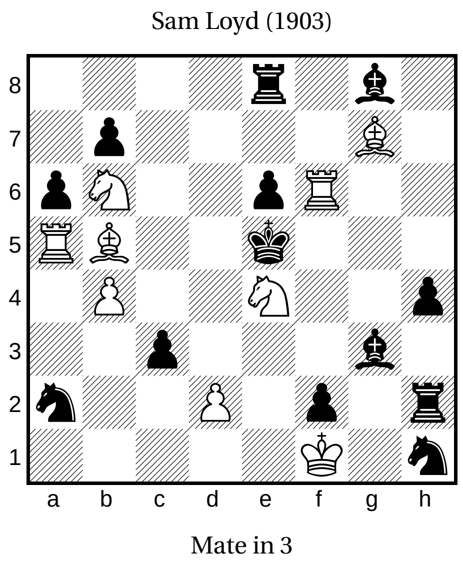
As we saw in the previous section, the problem of telling whether a Hamiltonian cycle exists has the property that finding solutions is hard—or so we believe—but checking them is easy. Another example of this phenomenon is factoring integers. As far as we know, there is no efficient way to factor a large integer N into its divisors—at least without a quantum computer—and we base the modern cryptosystems used
1.4 by intelligence agents and Internet merchants on this belief. On the other hand, given two numbers p and q it is easy to multiply them, and check whether $pq = N$.

This fact was illustrated beautifully at a meeting of the American Mathematical Society in 1903. The mathematician Frank Nelson Cole gave a "lecture without words," silently performing the multiplication

$$193\,707\,721 \times 761\,838\,257\,287 = 147\,573\,952\,588\,676\,412\,927$$

on the blackboard. The number on the right-hand side is $2^{67} - 1$, which the 17th-century French mathematician Marin Mersenne conjectured is prime. In 1876, Édouard Lucas managed to prove that it is composite, but gave no indication of what numbers would divide it. The audience, knowing full well how hard it is to factor 21-digit numbers, greeted Cole's presentation with a standing ovation. Cole later admitted that it had taken him "three years of Sundays" to find the factors.

On the other hand, there are problems for which even *checking* a solution is extremely hard. Consider the Chess problems shown in Figure 1.7. Each claims that White has a winning strategy, which will lead



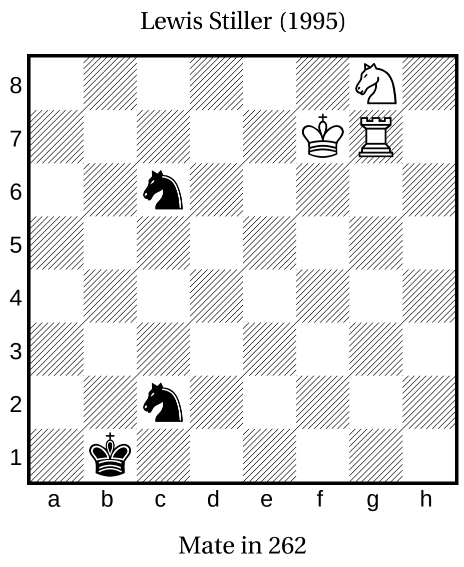


FIGURE 1.7: Chess problems are hard to solve—and hard to check.

inexorably to checkmate after n moves. On the left, $n = 3$, and seeing how to corner Black—after a very surprising first move—is the work of a pleasant afternoon. On the right, we have a somewhat larger value of n: we claim that White can force Black into checkmate after 262 moves.

But how, dear reader, can we prove this claim to you? Unlike the problems of the previous two sections, we are no longer playing solitaire: we have an opponent who will do their best to win. This means that it's not enough to prove the existence of a simple object like a Hamiltonian path. We have to show that there exists a move for White, such that no matter how Black replies, there exists a move for White, such that no matter how Black replies, and so on... until, at most 262 moves later, every possible game ends in checkmate for Black. As we go forward in time, our opponent's moves cause the game to branch into a exponential tree of possibilities, and we have to show that a checkmate awaits at every leaf. Thus a strategy is a much larger object, with a much deeper logical structure, than a path.

There is indeed a proof, consisting of a massive database of endgames generated by a computer search, that White can mate Black in 262 moves in the position of Figure 1.7. But verifying this proof far exceeds the capabilities of human beings, since it requires us to check every possible line of play. The best we can do is look at the program that performed the search, and convince ourselves that it will run correctly. As of 2007, an even larger search has confirmed the long-standing opinion of human players that Checkers is a draw under perfect play. For humans with our finite abilities, however, Chess and Checkers will always keep their secrets. 1.5

1.4 What Lies Ahead

As we have seen with our three examples, different problems require fundamentally different kinds of search, and different types of proof, to find or verify their solutions. Understanding how to solve problems as efficiently as possible—and understanding how, and why, some problems are extremely hard—is the subject of our book. In the chapters ahead, we will ask, and sometimes answer, questions like the following:

- Some problems have insights like Euler's, and others seem to require an exhaustive search. What makes the difference? What kinds of strategies can we use to skip or simplify this search, and for which problems do they work?
- A host of problems—finding Hamiltonian paths, coloring graphs, satisfying formulas, and balancing numbers—are all equally hard. If we could solve any of them efficiently, we could solve all of them. What do these problems have in common? How can we transform one of them into the other?
- If we could find Hamiltonian paths efficiently, we could also easily find short proofs—if they exist—of the great unsolved problems in mathematics, such as the Riemann Hypothesis. We believe that doing mathematics is harder than this, and that it requires all our creativity and intuition. But does it really? Can we prove that finding proofs is hard?
- Can one programming language, or kind of computer, solve problems that another can't? Or are all sufficiently powerful computers equivalent? Are even simple systems, made of counters, tiles, and billiard balls, capable of universal computation?
- Are there problems that no computer can solve, no matter how much time we give them? Are there mathematical truths that no axiomatic system can prove?
- If exact solutions are hard to find, can we find approximate ones? Are there problems where even approximate solutions are hard to find? Are there others that are hard to solve perfectly, but where we can find solutions that are as close as we like to the best possible one?
- What happens if we focus on the amount of memory a computation needs, rather than the time it takes? How much memory do we need to find our way through a maze, or find a winning strategy in a two-player game?
- If we commit ourselves to one problem-solving strategy, a clever adversary can come up with the hardest possible example. Can we defeat the adversary by acting unpredictably, and flipping coins to decide what to do?
- Suppose that Merlin has computational power beyond our wildest dreams, but that Arthur is a mere mortal. If Merlin knows that White has a winning strategy in Chess, can he convince Arthur of that fact, without playing a single game? How much can Arthur learn by asking Merlin random questions? What happens if Merlin tries to deceive Arthur, or if Arthur tries to "cheat" and learn more than he was supposed to?

- If flipping random coins helps us solve problems, do we need truly random coins? Are there strong pseudorandom generators—fast algorithms that produce strings of coin flips deterministically, but with no pattern that other fast algorithms can discover?
- Long proofs can have small mistakes hidden in them. But are there "holographic" proofs, which we can confirm are almost certainly correct by checking them in just a few places?
- Finding a solution to a problem is one thing. What happens if we want to generate a random solution, or count the number of solutions? If we take a random walk in the space of all possible solutions, how long will it take to reach an equilibrium where all solutions are equally likely?
- How rare are the truly hard examples of hard problems? If we make up random examples of a hard problem, are they hard or easy? When we add more and more constraints to a problem in a random way, do they make a sudden jump from solvable to unsolvable?
- Finally, how will quantum computers change the landscape of complexity? What problems can they solve faster than classical computers?

Problems

A great discovery solves a great problem, but there is a grain of discovery in the solution of any problem. Your problem may be modest, but if it challenges your curiosity and brings into play your inventive faculties, and if you solve it by your own means, you may experience the tension and enjoy the triumph of discovery.

George Pólya, *How To Solve It*

1.1 Handshakes. Prove that in any finite graph, the number of vertices with odd degree is even.

1.2 Pigeons and holes. Properly speaking, we should call our representation of Königsberg a *multigraph*, since some pairs of vertices are connected to each other by more than one edge. A *simple* graph is one in which there are no multiple edges, and no self-loops.

Show that in any finite simple graph with more than one vertex, there is at least one pair of vertices that have the same degree. Hint: if n pigeons try to nest in $n-1$ holes, at least one hole will contain more than one pigeon. This simple but important observation is called the *pigeonhole principle*.

1.3 Proving Euler's claim. Euler didn't actually prove that having vertices with even degree is sufficient for a connected graph to be Eulerian—he simply stated that it is obvious. This lack of rigor was common among 18th century mathematicians. The first real proof was given by Carl Hierholzer more than 100 years later. To reconstruct it, first show that if every vertex has even degree, we can cover the graph with a set of cycles such that every edge appears exactly once. Then consider combining cycles with moves like those in Figure 1.8.

1.4 Finding an Eulerian path. Let's turn the proof of the previous problem into a simple algorithm that constructs an Eulerian path. If removing an edge will cause a connected graph to fall apart into two pieces, we call that edge a *bridge*. Now consider the following simple rule, known as Fleury's algorithm: at each step, consider the graph G' formed by the edges you have not yet crossed, and only cross a bridge of G' if you have to. Show that if a connected graph has two vertices of odd degree and we start at one of them, this algorithm will produce an Eulerian path, and that if all vertices have even degree, it will produce an Eulerian cycle no matter where we start.

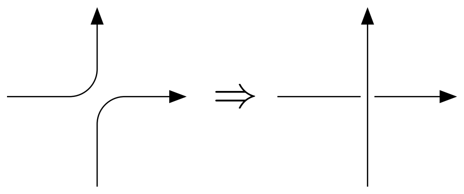


FIGURE 1.8: Combining cycles at a crossing.

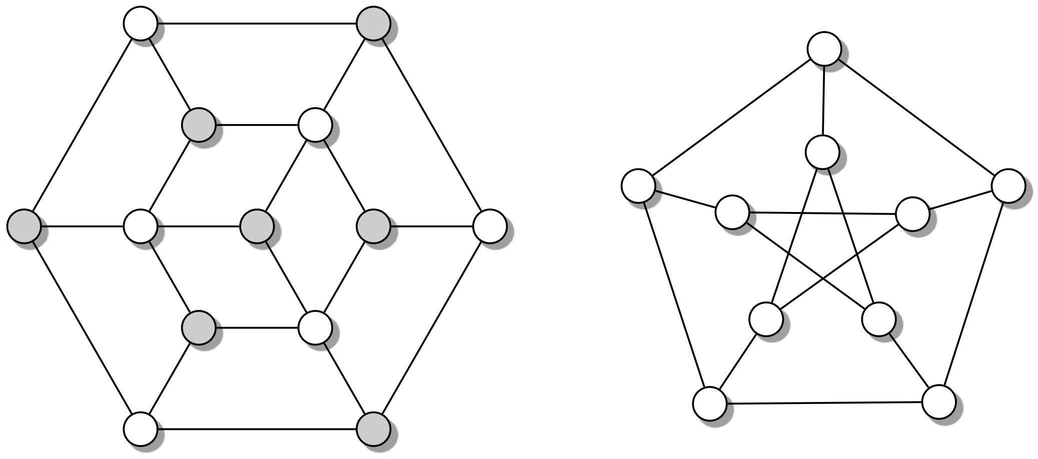


FIGURE 1.9: Two graphs that have Hamiltonian paths, but not Hamiltonian cycles.

1.5 One-way bridges. A *directed graph* is one where each edge has an arrow on it. Thus there can be an edge from u to v without one from v to u, and a given vertex can have both incoming and outgoing edges. An Eulerian path would be one that crosses each edge once, moving in the direction allowed by the arrow. Generalize Euler's theorem by stating under what circumstances a directed graph is Eulerian.

1.6 Plato and Hamilton. Inspired by Hamilton's choice of the dodecahedron, consider the other four Platonic solids, and the graphs consisting of their corners and edges: the tetrahedron, cube, octahedron and icosahedron. Which ones are Hamiltonian? Which are Eulerian?

1.7 A rook's tour. Let G be an $m \times n$ grid—that is, a graph with mn vertices arranged in an $m \times n$ rectangle, with each vertex connected to its nearest neighbors. Assume that $m, n > 1$. Prove that G is Hamiltonian if either m or n is even, but not if both m and n are odd.

1.8 Symmetry and parity. Show that each of the graphs in Figure 1.9 has a Hamiltonian path, but no Hamiltonian cycle. Hint: there is an easy proof for the one on the left. For the one on the right, which is called the Petersen graph, try to exploit its symmetry.

1.9 The Chinese postman. The *Chinese postman problem*, named in honor of the Chinese mathematician Mei-Ko Kwan, asks for the shortest cyclic tour of a graph that crosses every edge *at least* once. Show that for any connected graph, there is a tour that crosses every edge at most twice, and show that this worst case only happens if the graph is a tree. If you want a real challenge, try to devise an algorithm that solves this problem in polynomial time.

1.10 Existence, search, and the oracle. We can consider two rather different problems regarding Hamiltonian cycles. One is the *decision problem*, the yes-or-no question of whether such a cycle exists. The other is the *search problem* or *function problem*, in which we want to actually find the cycle. However, the hardness of these problems is very closely related.

Suppose that there is an oracle in a nearby cave. She will tell us, for the price of one drachma per question, whether a graph has a Hamiltonian cycle. If it does, show that by asking her a series of questions, perhaps involving modified versions of the original graph, we can find the Hamiltonian cycle after spending a number of drachmas that grows polynomially as a function of the number of vertices. Thus if we can solve the decision problem in polynomial time, we can solve the search problem as well.

Notes

1.1 Graph theory. Euler's paper on the Königsberg bridges [269] can be regarded as the birth of graph theory, the mathematics of connectivity. The translation we use here appears in [111], which contains many of the key early papers in this field. Today graph theory is a very lively branch of discrete mathematics with many applications in chemistry, engineering, physics, and computer science.

We will introduce concepts and results from graph theory "on the fly," as we need them. For an intuitive introduction, we recommend Trudeau's little treatise [781]. For a more standard textbook, see Bollobás [119], and if you still have questions you should consult the *Handbook on Graph Theory* [348].

Hierholzer's proof, which we ask you to reconstruct in Problem 1.3, appeared in [392]. The Chinese Postman of Problem 1.9 was studied by Mei-Ko Kwan in 1962 [503].

The river Pregel still crosses the city of Kaliningrad, but the number of bridges and their topology changed considerably during World War II.

1.2 Persuasive proofs. According to the great Hungarian mathematician Paul Erdős, God has a book that contains short and insightful proofs of every theorem, and sometimes humans can catch a glimpse of a few of its pages. One of the highest forms of praise that one mathematician can give another is to say "Ah, you have found the proof from the book." [27].

In a sense, this is the difference between Eulerian and Hamiltonian paths. If a large graph G lacks an Eulerian path, Euler's argument gives a "book proof" of that fact. In contrast, as far as we know, the only way to prove that G lacks a Hamiltonian path is an ugly exhaustive search.

This dichotomy exists in other areas of mathematics as well. A famous example is Thomas Hales' proof of the Kepler conjecture [356], a 400-year-old claim about the densest packings of spheres in three-dimensional space. After four years of work, a group of twelve referees concluded that they were "99% certain" of the correctness of the proof. Hales' proof relies on exhaustive case checking by a computer, a technique first used in the proof of the Four Color Theorem. Such a proof may confirm that a fact is true, but it offers very little illumination about *why* it is true. See [219] for a thought-provoking discussion of the complexity of mathematical proofs, and how computer-assisted proofs will change the nature of mathematics.

1.3 A knight's tour. One instance of the Hamiltonian path problem is far older than the Icosian game. Around the year 840 A.D., a Chess player named al-Adli ar-Rumi constructed a *knight's tour* of the chessboard, shown in Figure 1.10. We can think of this as a Hamiltonian path on the graph whose vertices are squares, and where two vertices are adjacent if a knight could move from one to the other. Another early devotee of knight's tours was the Kashmiri poet Rudrata (circa 900 A.D.). In their lovely book *Algorithms* [215], Dasgupta, Papadimitriou and Vazirani suggest that Hamiltonian paths be called Rudrata paths in his honor.

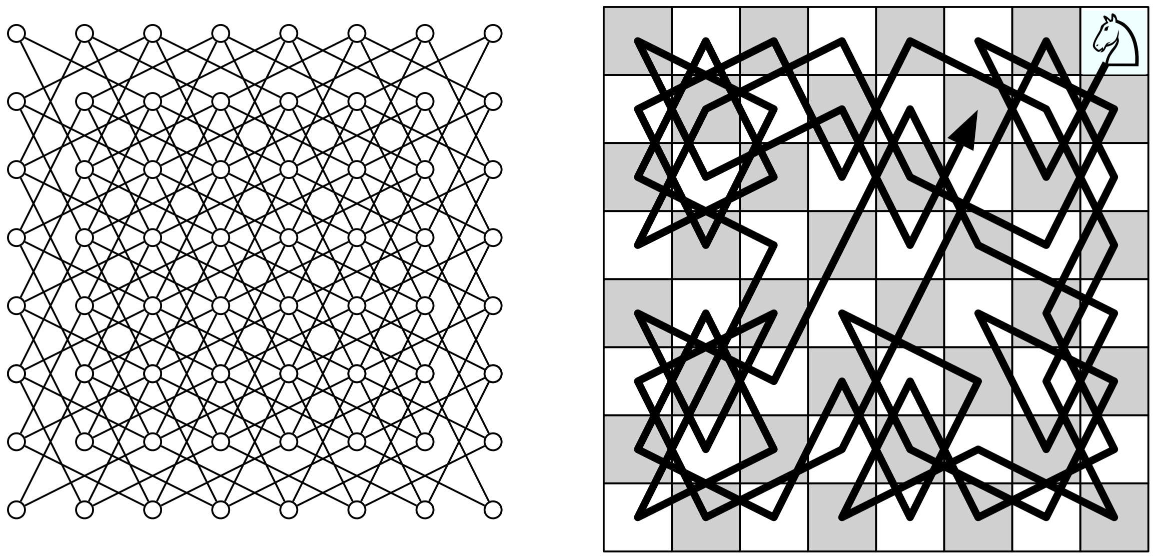


FIGURE 1.10: On the left, the graph corresponding to the squares of the chessboard with knight's moves between them. On the right, Al-Adli's Hamiltonian tour, or knight's tour, of this graph. Note that with one more move it becomes a Hamiltonian cycle.

1.4 Cryptic factors. As we will see in Chapter 15, it is not quite right to say that cryptosystems like RSA public-key encryption are based on the hardness of factoring. Rather they are based on other number-theoretic problems, which are conceivably easier than factoring. However, it is true that if we can factor large integers efficiently, then we can break these cryptosystems too.

1.5 Endgame. The *endgame tablebase* is a computerized database of all endgame positions in Chess with a given number of pieces, that reveals the value (win, loss or draw) of each position and and how many moves it will take to achieve that result with perfect play [774]. The 262-move problem in Figure 1.7 was found by Stiller [759], who extended the tablebase to six-piece endgames. Of all positions with the given material that are a win, it has the longest "distance to mate."

The proof that Checkers is a draw [715] uses a similar endgame database that contains the value of all Checkers positions with ten pieces or fewer. There are 3.9×10^{13} of these positions, compressed into 237 gigabytes of diskspace. The proof traces all relevant lines of play from the starting position to one of the positions in this database. This proof constitutes what game theorists call a *weak* solution of a game. For a *strong* solution, we would have to compute the optimal move for every legal position on the board, not just the opening position. We will discuss the enormous difficulty of this problem in Chapter 8.

Can we hope for a weak solution of Chess? The six-piece endgame database for Chess is more than five times bigger than the ten-piece database for Checkers. This is no surprise since Checkers uses only half of the squares of the 8×8 board and has only two types of pieces (man and king). In addition, the rules of Checkers, in which pieces can only move forwards and captures are forced, keep the tree from branching too quickly. For Chess, even a weak solution seems out of reach for the foreseeable future.