1 A Path to Understanding Computation

What is computation? This question lies at the core of computer science. This chapter provides an answer—at least a tentative one—and connects the notion of computation to some closely related concepts. In particular, I explain the relationship between computation and the concepts of problem solving and algorithms. To this end, I describe two complementary aspects of computation: what it does, and what it is.

The first view, *computation solves problems*, emphasizes that a problem can be solved through computation once it is suitably represented and broken down into subproblems. It not only reflects the tremendous impact computer science has had in so many different areas of society but also explains why computation is an essential part of all kinds of human activities, independent of the use of computing machines.

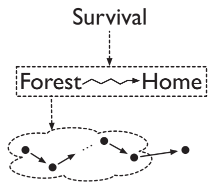
However, the problem-solving perspective leaves out some important aspects of computation. A closer look at the differences between computation and problem solving leads to a second view, *computation is algorithm execution*. An algorithm is a precise description of computation and makes it possible to automate and analyze computation. This view portrays computation as a process consisting of several steps, which helps explain how and why it is so effective in solving problems.

The key to harnessing computation lies in grouping similar problems into one class and designing an algorithm that solves each problem in that class. This makes an algorithm similar to a skill. A skill such as baking a cake or repairing a car can be invoked at different times and thus can be employed repeatedly to solve different instances of a particular problem class. Skills can also be taught to and shared with others, which gives them an even wider impact. Similarly, we can execute an algorithm repeatedly for different problem instances and generate with each execution a computation that solves the problem at hand.

Dividing Problems into Triviality

Let us start with the first perspective and consider computation as a process that solves a specific problem. As an example, I use the well-known story of Hansel and Gretel, who were left to die in the woods by their parents. Let's examine Hansel's clever idea that allowed him and Gretel to find their way back home after being left behind in the forest. The story unfolds in the context of a famine, when Hansel and Gretel's stepmother urges their father to lead the children into the forest and abandon them, so that the parents can survive. Having overheard his parents' conversation, Hansel goes outside later that night and collects several handfuls of small pebbles that he stuffs into his pockets. The next day, during their walk into the forest, he drops the pebbles along the way as markers for the way back home. After the parents have left them, the children wait until it is dark and the pebbles begin to shine in the moonlight. They then follow the pebbles until they return home.

The story doesn't end here, but this part provides us with a peculiar example of how a problem is solved using computation. The problem to be solved is one of survival—certainly much more serious than the problem of getting up. The survival problem presents itself as a task of moving from a location in the forest to the location of Hansel and Gretel's home. This is a nontrivial problem particularly because it cannot be solved in one step. A problem that is too complex to be solved in one step has to be broken down into subproblems that are easy to solve and whose solutions can be combined into a solution for the overall problem.



The problem of finding the way out of the forest can be decomposed by identifying a sequence of intermediate locations that are close enough to each other that one can easily move between them. These locations form a path out of the forest back to Hansel and Gretel's home, and the individual movements from one location to the next are easy to achieve. When combined, they yield a movement from the starting location in the forest to the home. This movement solves Hansel and Gretel's problem of survival in a systematic way. Systematic problem solving is one key characteristic of computation.

As this example illustrates, a computation usually consists of not just one but many steps. Each of these steps solves a subproblem and changes the problem situation a little bit. For example, each move by Hansel and Gretel to the next pebble is a step in the computation that changes their position in the forest, which corresponds to solving the subproblem of reaching the next target on the path home. While in most cases each individual step will bring the computation closer to the solution, this does not necessarily have to be the case for every step. Only all steps taken together have to yield the solution. In the story, while each position that Hansel and Gretel go through will generally be closer to home, it is also likely that the path is not a straight line. Some pebbles may even cause detours, for example, to move around obstacles or to cross a river using a bridge, but this does not change the effect of the combined movement.

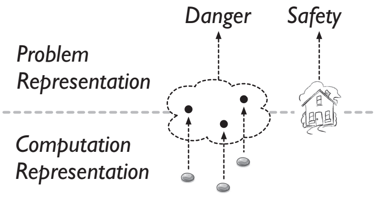
The important lesson is that a solution is obtained through a systematic problem decomposition. While decomposition is a key strategy to obtaining a solution to a problem, it is not always sufficient by itself, and solutions may depend on supplementary items—in the case of Hansel and Gretel, the pebbles.

No Computation without Representation

If a computation consists of a number of steps, what does each of these steps actually do, and how can all the steps together produce a solution to the given problem? To produce an aggregate effect, each step has to have an effect that the next steps can build on so that the cumulative effect produced by all the steps results in a solution for the problem. In the story the effect of each step is to change Hansel and Gretel's location, and the problem is solved when the location is finally changed to their home. In general, a step in a computation can have an effect on almost anything, be it concrete physical objects or abstract mathematical entities.

To solve a problem it is necessary that a computation manipulate a *representation* of something meaningful in the real world. Hansel and Gretel's locations represent one of two possible states: all locations in the forest represent the problem state of danger and possibly death, while their home represents the solution state of safety and survival. This is why the computation that brings Hansel and Gretel home solves a problem—it moves them from danger to safety. In contrast, a computation that leads from one place in the forest to another would not achieve that.

This example has another level of representation. Since the computation that is defined by moves between locations is carried out by Hansel and Gretel, the locations must be recognizable to them, which is why Hansel drops the pebbles along the way. The pebbles represent the locations in a form that enables the



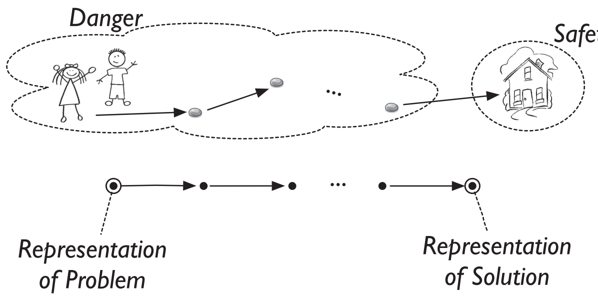


Figure 1.1 Computation is a process for solving a particular problem. Usually, a computation consists of several steps. Starting with a representation of the problem, each step transforms the representation until the solution is obtained. Hansel and Gretel solve the problem of surviving through a process of changing their position step-by-step and pebble-by-pebble from within the forest to their home.

computer, that is, Hansel and Gretel, to actually carry out the steps of the computation. It is common to have several layers of representation. In this case we have one that defines the problem (locations) and one that makes it possible to compute a solution (pebbles). In addition, all the pebbles taken together constitute another level of representation, since they represent the path out of the forest back home. These representations are summarized in table 1.1.

Figure 1.1 summarizes the problem-solving picture of computation; it shows Hansel and Gretel's way-finding as an instance of the view that computation manipulates representation in a sequence of steps. In the getting-up problem we also find representations, for example, as location (in bed, out of bed) and as an alarm clock representing time. Representations can take many different forms. They are discussed in more detail in chapter 3.

Table 1.1

Computation Representation		*Problem Representation*	
Object	**Represents**	**Concept**	**Represents**
One pebble	Location in forest	Location in forest	Danger
	Home	Home	Safety
All pebbles	Path out of forest	Path out of forest	Problem Solution

Beyond Problem Solving

Regarding computation as a problem-solving process captures the purpose of computation but doesn't explain what computation really *is*. Moreover, the problem-solving view has some limitations, since not every act of problem solving is a computation.

As illustrated in figure 1.2, there are computations, and there is problem solving. Although these often overlap, some computations do not solve problems, and some problems are not solved through computation. The emphasis in this book is on the intersection of computations and problem solving, but to make this focus clear, I consider some examples for the other two cases.

For the first case, imagine a computation consisting of following pebbles from one place to another within the forest. The steps of this process are in principle the same as in the original story, but the corresponding change of location would not solve Hansel and Gretel's problem of survival. As an even more drastic example, imagine the situation when the pebbles are arranged in a loop, which means the corresponding computation doesn't seem to achieve anything, since the initial and final positions are identical. In other words, the computation has no cumulative effect. The difference between these two cases and the one in the story is the meaning that is attached to the process.

Processes without such an apparent meaning still qualify as computation, but they may not be perceived as problem solving. This case is not hugely important, since we can always assign some arbitrary meaning to the representation operated on by a particular computation. Therefore, any computation could arguably be considered problem solving; it always depends on what meaning is associated with the representation. For example, following a loop inside a forest may not be helpful for Hansel and Gretel, but it could solve the problem of getting exercise for a runner. Thus, whether a computation solves a problem is in the eye of the beholder, that is, in the utility of a computation. In any case, whether or not one grants a particular computation the status of problem solving does not affect the essence of computation.

The situation is markedly different for noncomputational problem solving, since it provides us with further criteria for computation. In figure 1.2 two such criteria are mentioned, both of which are, in fact, very closely related. First, if a problem is solved in an ad hoc way that doesn't follow a particular method, it is not a computation. In other words, a computation has to be systematic. We can find several instances of this kind of noncomputational problem solving in the story. One example occurs when Hansel and Gretel are held captive by the witch who tries to fatten up Hansel with the goal of eating him. Since she can't see well, she estimates Hansel's weight by feeling his finger. Hansel misleads the witch about his weight by using a little bone in place of his

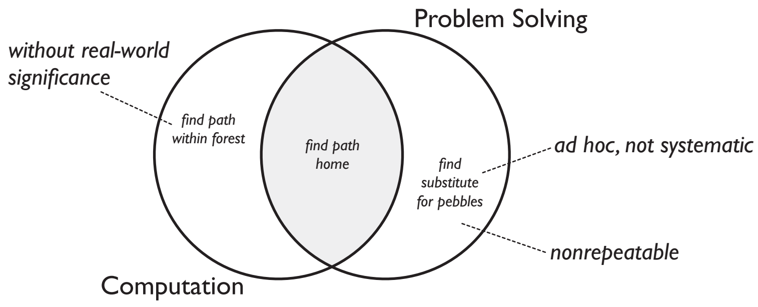


Figure 1.2 Distinguishing problem solving from computation. A computation whose effect carries no meaning in the real world does not solve any problems. An ad hoc solution to a problem that is not repeatable is not a computation.

finger. This idea is not the result of a systematic computation, but it solves the problem: it postpones the witch's eating Hansel.

Another example of noncomputational problem solving occurs right after Hansel and Gretel have returned home. The parents plan to lead them into the forest again the next day, but this time the stepmother locks the doors the night before to prevent Hansel from collecting any pebbles. The problem is that Hansel cannot get access to the pebbles that served him so well the first time and that he relied on for finding the way back home. His solution is to find a substitute in the form of breadcrumbs. The important point here is how Hansel arrives at this solution—he has an idea, a creative thought. A solution that involves a eureka moment is generally very difficult, if not impossible, to derive systematically through a computation because it requires reasoning on a subtle level about objects and their properties.

Unfortunately for Hansel and Gretel, the breadcrumbs solution doesn't work as expected:

> *When the moon came, they set out, but they found no crumbs, for the many thousands of birds which fly about in the woods and fields had picked them all up.*[1]

Since the breadcrumbs are gone, Hansel and Gretel can't find their way home, and the rest of the story unfolds.

However, let us assume for a moment that Hansel and Gretel could somehow have found their way back home again and that the parents would have tried a third time to leave them behind in the forest. Hansel and Gretel would have had to think of yet another means to mark their way home. They would have had to find something else to drop along the way, or maybe they would have tried to mark trees or bushes. Whatever the solution, it would have been produced by thinking about the problem and having another creative idea, but not by systematically applying a method. This highlights the other criterion for computation, which is its ability to be repeated and solve many similar problems. The method for solving the way-finding problem by following pebbles is different in this regard because it can be executed repeatedly for many different pebble placements.

To summarize, while the problem-solving perspective shows that computation is a systematic and decomposable process, it is not sufficient to give a comprehensive and precise picture of computation. Viewing computation as problem solving demonstrates how computation can be employed in all kinds of situations and thus illustrates its importance but ignores some important attributes that explain how computation works and why it can be successfully applied in so many different ways.

When Problems Reappear

Hansel and Gretel are faced with the problem of finding their way back home *twice*. Except for the practical problems caused by the lack of pebbles, the second instance could be solved in the same way as the first, namely, by following a series of markers. There is nothing really surprising about this fact, since Hansel and Gretel are simply applying a general method of way-finding. Such a method is called an *algorithm*.

Let us take a look at the algorithm that was used by Hansel and Gretel to find their way home. The exact method is not explained in detail in the original fairy tale. All we are told is the following:

> *And when the full moon had risen, Hansel took his little sister by the hand, and followed the pebbles which shone like newly-coined silver pieces, and showed them the way.*

A simple algorithm that fits this characterization is given, for example, by the following description:

> Find a shining pebble that was not visited before, and go toward it.
> Continue this process until you reach your parents' house.

An important property of an algorithm is that it can be used repeatedly by the same or a different person to solve the same or a closely related problem. An algorithm that generates a computation with a physical effect is useful even if it solves only one specific problem. For example, a recipe for a cake will produce the same cake over and over again. Since the output of the algorithm is transient—the cake gets eaten—reproducing the same result is very useful. The same holds for the problem of getting out of bed and dressed; the effect of the algorithm has to be reproduced every day, although likely with different clothes and at a different time on weekends. This also applies to Hansel and Gretel. Even if they are brought to the same place in the forest as on the first day, getting home has to be recomputed by repeating the algorithm to solve the exact same problem.

The situation is different for algorithms that produce nonphysical, abstract results, such as numbers. In such a case one can simply write down the result and look it up the next time it is needed instead of executing the algorithm again. For an algorithm to be useful in such situations it must be able to solve a whole class of problems, which means that it must be possible to apply the method to several different, but related, problems.[2]

In the story the method is general enough to solve many different way-finding problems, since the exact positions of the pebbles do not matter. No matter where exactly in the forest the parents lead the children to, the algorithm will work in every case[3] and consequently will cause a computation that solves Hansel and Gretel's survival problem. Much of the power and impact of algorithms comes from the fact that *one* algorithm gives rise to *many* computations.

The notion of an algorithm is one of the most important concepts in computer science because it provides the foundation for the systematic study of computation. Accordingly, many different aspects of algorithms are discussed throughout this book.

Do You Speak "Algorithmish"?

An algorithm is a description of how to perform a computation and must therefore be formulated in some language. In the story the algorithm is only marginally mentioned. Hansel certainly has the algorithm in his head and might have told Gretel about it, but the algorithm is not written down as part of the story. However, the fact that an algorithm can be written down is an important property because it enables the reliable sharing of the algorithm and thus allows many people to use it to solve problems. The ability to express an algorithm in some language supports the proliferation of computation because instead of one person producing many computations, it facilitates many people's producing even more computations. If the language in which the algorithm is expressed can be understood by a computing