

# Problem Solving and Algorithms

## Problems, Solutions, and Tools

I have a problem! I need to thank my friend for the birthday present he sent me. I could send a thank you note through the mail. I could call him on the telephone. I could send him an email message. I could drive to his house and thank him in person. In fact, there are many ways I could thank him, but that's not the point. The point is that I must decide how I want to solve the problem, and use the appropriate tool to implement (carry out) my plan. The postal service, the telephone, the internet, and my automobile are tools that I can use, but none of these actually solves my problem. In a similar way, a computer does not solve problems, it's just a tool that I can use to implement my plan for solving the problem.

Knowing that My friend appreciates creative and unusual things, I have decided to hire a singing messenger to deliver my thanks. In this context, the messenger is a tool, but one that needs instructions from me. I have to tell the messenger where My friend lives, what time I would like the message to be delivered, and what lyrics I want sung. A computer program is similar to my instructions to the messenger.

The story of My friend uses a familiar context to set the stage for a useful point of view concerning computers and computer programs. The following list summarizes the key aspects of this point of view.

- A computer is a tool that can be used to implement a plan for solving a problem.
- A computer program is a set of instructions for a computer. These instructions describe the steps that the computer must follow to implement a plan.
- An algorithm is a plan for solving a problem.
- A person must design an algorithm.
- A person must translate an algorithm into a computer program.

This point of view sets the stage for a process that we will use to develop solutions to Jeroo problems. The basic process is important because it can be used to solve a wide variety of problems, including ones where the solution will be written in some other programming language.

## An Algorithm Development Process

Every problem solution starts with a plan. That plan is called an algorithm.

An algorithm is a plan for solving a problem.

There are many ways to write an algorithm. Some are very informal, some are quite formal and mathematical in nature, and some are quite graphical. The instructions for connecting a DVD player to a television are an algorithm. A mathematical formula such as  $\pi R^2$  is a special case of an algorithm. The form is not particularly important as long as it provides a good way to describe and check the logic of the plan.

The development of an algorithm (a plan) is a key step in solving a problem. Once we have an algorithm, we can translate it into a computer program in some programming language. Our algorithm development process consists of five major steps.

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

### **Step 1: Obtain a description of the problem.**

This step is much more difficult than it appears. In the following discussion, the word client refers to someone who wants to find a solution to a problem, and the word developer refers to someone who finds a way to solve the problem. The developer must create an algorithm that will solve the client's problem.

The client is responsible for creating a description of the problem, but this is often the weakest part of the process. It's quite common for a problem description to suffer from one or more of the following types of defects: (1) the description relies on unstated assumptions, (2) the description is ambiguous, (3) the description is incomplete, or (4) the description has internal contradictions. These defects are seldom due to carelessness by the client. Instead, they are due to the fact that natural languages (English, French, Korean, etc.) are rather imprecise. Part of the developer's responsibility is to identify defects in the description of a problem, and to work with the client to remedy those defects.

### **Step 2: Analyze the problem.**

The purpose of this step is to determine both the starting and ending points for solving the problem. This process is analogous to a mathematician determining what is given and what must be proven. A good problem description makes it easier to perform this step.

When determining the starting point, we should start by seeking answers to the following questions:

What data are available?

Where is that data?

What formulas pertain to the problem?

What rules exist for working with the data?

What relationships exist among the data values?

When determining the ending point, we need to describe the characteristics of a solution. In other words, how will we know when we're done? Asking the following questions often helps to determine the ending point.

What new facts will we have?

What items will have changed?

What changes will have been made to those items?

What things will no longer exist?

### **Step 3: Develop a high-level algorithm.**

An algorithm is a plan for solving a problem, but plans come in several levels of detail. It's usually better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later. We can use an everyday example to demonstrate a high-level algorithm.

Problem: I need to send a birthday card to my brother, Mark.

Analysis: I don't have a card. I prefer to buy a card rather than make one myself.

High-level algorithm:

Go to a store that sells greeting cards

Select a card

Purchase a card

Mail the card

This algorithm is satisfactory for daily use, but it lacks details that would have to be added were a computer to carry out the solution. These details include answers to questions such as the following.

"Which store will I visit?"

"How will I get there: walk, drive, ride my bicycle, take the bus?"

"What kind of card does Mark like: humorous, sentimental, risqué?"

These kinds of details are considered in the next step of our process.

### **Step 4: Refine the algorithm by adding more detail.**

A high-level algorithm shows the major steps that need to be followed to solve a problem. Now we need to add details to these steps, but how much detail should we add? Unfortunately, the answer to this question depends on the situation. We have to consider who (or what) is going to implement the algorithm and how much that person (or thing) already knows how to do. If someone is going to purchase Mark's birthday card on my behalf, my instructions have to be adapted to whether or not that person is familiar with the stores in the community and how well the purchaser knows my brother's taste in greeting cards.

When our goal is to develop algorithms that will lead to computer programs, we need to consider the capabilities of the computer and provide enough detail so that someone else could use our algorithm to write a computer program that follows the steps in our algorithm. As with the birthday card problem, we need to adjust the level of detail to match the ability

of the programmer. When in doubt, or when you are learning, it is better to have too much detail than to have too little.

Most of our examples will move from a high-level to a detailed algorithm in a single step, but this is not always reasonable. For larger, more complex problems, it is common to go through this process several times, developing intermediate level algorithms as we go. Each time, we add more detail to the previous algorithm, stopping when we see no benefit to further refinement. This technique of gradually working from a high-level to a detailed algorithm is often called stepwise refinement.

Stepwise refinement is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.

### **Step 5: Review the algorithm.**

The final step is to review the algorithm. What are we looking for? First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem. Once we are satisfied that the algorithm does provide a solution to the problem, we start to look for other things. The following questions are typical of ones that should be asked whenever we review an algorithm. Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

Does this algorithm solve a very specific problem or does it solve a more general problem? If it solves a very specific problem, should it be generalized?

For example, an algorithm that computes the area of a circle having radius 5.2 meters (formula  $\pi * 5.2^2$ ) solves a very specific problem, but an algorithm that computes the area of any circle (formula  $\pi * R^2$ ) solves a more general problem.

Can this algorithm be simplified?

One formula for computing the perimeter of a rectangle is:

length + width + length + width

A simpler formula would be:

$2.0 * (\text{length} + \text{width})$

Is this solution similar to the solution to another problem? How are they alike? How are they different?

For example, consider the following two formulae:

Rectangle area = length \* width

Triangle area =  $0.5 * \text{base} * \text{height}$

Similarities: Each computes an area. Each multiplies two measurements.

Differences: Different measurements are used. The triangle formula contains 0.5.

Hypothesis: Perhaps every area formula involves multiplying two measurements.