

Algorithms: the Foundation of Systematic Problem-Solving

"*algorithm*, a finite collection of simple instructions that can be performed by a computer and is guaranteed to halt in a finite amount of time."

source: the Analytical Engine, p. 26

QUESTION: Suppose a program is *designed* not to halt in a finite amount of time? Is it implementing an algorithm?

Algorithms: the Foundation of Systematic Problem-Solving

"algorithm A rigid mathematical (or logical) relationship which has only one starting point and one finishing point. ... Each algorithm must have a finite list of executable instructions which must eventually come to an end.

"The British mathematician Alan Turing proved [1936] that an algorithmic approach can be used to solve any mathematical or logical problem for which a solution is known to exist. This means that one can deduce that any solvable problem can be handled by a computer, provided the correct algorithm is used.

"In computing, an algorithm is the plan, routine, or process of defining a set of instructions so that a computer program can be written to find the answer / solution to a given problem or task."

source: Prentice-Hall's Illustrated Dictionary of Computing (3rd ed.) (Prentice-Hall, 1998), p. 19

Developing an Algorithm

Problem Specification. You are driving a car with two friends and suddenly get a flat tire. Fortunately, there is a spare tire and jack in the trunk.

Analysis. After pulling over to the side of the road, you might decide to subdivide the problem of changing a tire into the subproblems below.

Design. Here are the main steps in an algorithm to change a tire.

Algorithm.

1. Loosen the lug nuts on the flat tire; don't remove them yet.
2. Get the jack and jack up the car.
3. Remove the lug nuts from the flat tire and remove the tire.
4. Get the spare tire, place it on the wheel, and tighten the nuts.
5. Lower the car.
6. Secure the jack and flat tire in the trunk.

Because these steps are relatively independent, you might decide to assign subproblem 1 to friend A, subproblem 2 to friend B, subproblem 3 to yourself, and so on. If friend B has used a jack before, the whole process should proceed smoothly; however, if friend B does not know how to use a jack, you need to refine step 2 further.

Step 2 Refinement

- 2.1. Get the jack from the trunk.
- 2.2. Place the jack under the car near the flat tire.
- 2.3. Insert the jack handle in the jack.
- 2.4. Place a block of wood under the car to keep it from rolling.
- 2.5. Jack up the car until there is enough room for the spare tire.

Step 2.4 requires a bit of decision making on your friend's part. Because the actual placement of the block of wood depends on whether the car is facing uphill or downhill, friend B needs to refine step 2.4.

Step 2.4 Refinement

2.4.1 If the car is facing uphill, then place the block of wood in back of a tire that is not flat; if the car is facing downhill, then place the block of wood in front of a tire that is not flat.

This is actually a conditional action: One of two alternative actions is executed, depending on a certain condition.

2.4.1 If the car is facing uphill, then place the block of wood in back of a tire that is not flat.

2.4.2 If the car is facing downhill, then place the block of wood in front of a tire that is not flat.

QUESTION: What if the car is on flat ground?

Finally, step 2.5 involves a repetitive action: moving the jack handle until there is sufficient room to put on the spare tire. Often, people stop when the car is high enough to remove the flat tire, forgetting that an inflated tire requires more room. It may take a few attempts to complete step 2.5.

Step 2.5 Refinement.

2.5.1. Move the jack handle repeatedly until the car is high enough off the ground that the spare tire can be put on the wheel.

The Power of Algorithms

- Algorithms can be written to solve a multitude of (sometimes very complex) problems
- Yet they are made up of careful combinations of *very simple* operations
- Simple operations are combined in the following ways
 - *sequence* - one operation (or group of operations) directly follows another
 - *condition (selection)* - an operation (or group of operations) is executed, or not executed, based on the results of a yes/no (true/false) decision
 - *repetition (loop)* - an operation (or group of operations) is executed repeatedly, a given number of times, or until a given condition is true
- Believe it or not, that's all! *All* algorithms can be constructed this way! That this is so was *proved* in 1966 (Böhm-Jacopini).

The Power of Algorithms

- Algorithms can be written to solve a multitude of (sometimes very complex) problems
- Yet they are made up of careful combinations of *very simple* operations
- Simple operations are combined in the following ways
 - *sequence* - one operation (or group of operations) directly follows another
 - *selection (condition)* - an operation (or group of operations) is executed, or not executed, based on the results of a yes/no (true/false) decision
 - *repetition (loop)* - an operation (or group of operations) is executed repeatedly, a given number of times, or until a given condition is true
- Believe it or not, that's all! *All* algorithms can be constructed this way! That this is so was *proved* in 1966 (Böhm-Jacopini).

Unit Price of a Pizza

Problem

You and your college roommates frequently order a late-night pizza snack. There are many pizzerias in the area that deliver to dormitories. Because you are on a tight budget, you would like to know which size pizza is the best value.

Analysis

To find which pizza is the best value, we must be able to do a meaningful comparison of pizza costs. Let's use *unit price*.

Pizzas are sold by size (diameter), measured in inches, so we can define the unit price of a pizza as its price divided by its area (that is, dollars per square inch).

The pizza problem

Initial Algorithm for Solution to Pizza Problem

1. For each size of pizza, read in the pizza size and price and compute unit cost. Compare the unit cost just computed with the previous unit costs and save the size and price of the pizza whose unit cost is the smallest so far.
2. Display the size and price of the pizza with the smallest unit cost.

The purpose of step 1 of the algorithm is to perform the cost computation for each individual pizza and somehow save the size and price of the pizza whose unit cost was the smallest. After all costs are computed, step 2 displays the size and price of the pizza that is the best buy.

Step 1 Refinement

- 1 *Repeat* the following steps for each size of pizza:
 - 1.1. Read in the next pizza size and price.
 - 1.2. Compute the unit price.
 - 1.3. *If* the new unit price is the smallest one so far, *then* save this pizza's size, price, and unit price.

Step 1 specifies the *repetition* of a group of steps: step 1.1 (read), step 1.2 (compute), and step 1.3 (compare). We will *repeat* these steps as many times as necessary until all unit prices are computed.

Each time we compute a new unit price, step 1.3 compares it to the others, and the current pizza's size and price are saved *if* its unit price is smaller than any others computed so far. *If* the unit price is not the smallest so far, the current pizza's size and price are not saved.

Step 1.3 is a selection step because it selects between the two possible outcomes: (a) save the pizza's data and (b) do not save the pizza's data.