MODIFYING ALGORITHMS: Generalizing, Extending, Foolproofing, Embedding

Algorithms often go through many changes in their "lifetimes." Sometimes they are made more powerful, more useful, more convenient, more efficient, or more foolproof. Sometimes they are used as parts of larger algorithms.

Generalizing algorithms is the process of making them apply to more cases. For example, the previous pay algorithm, repeated in figure 1, applies only to people making the same constant rate of 10 dollars an hour. This could be modified by allowing the input of any rate, say r, in addition to the hours h. This modification, shown in figure 2, now applies to more people, working at any rate, and is said to be more general. Actually the overtime value of 40, which appears in three places, could be replaced by a symbolic B (for break point), so becoming even more general. Also, the values are real numbers, with decimals, as opposed to the previous whole or integer numbers.

Extending algorithms to include more cases is also very common. For example, the original algorithm pays an overtime rate (time and a half) for any hours greater than 40. Often after more hours are worked (usually 60) there is a greater overtime rate of twice the regular rate. This extension of the original algorithm is shown in figure 3. Let us execute or "trace" this algorithm for an input value of 100 hours. First h is input, then it is compared to 60 and the rightmost path is taken out of the decision box into an action of computing.

```
p = 700 + 20*(h - 60) = 700 + 20*40 = 700 + 800 = $1500.
```

This general formula (and others) can be derived from finding the area of various rectangles under the graph of r vs h, an extension of the previous pay problem. Notice that by doubling the number of hours worked (from 50 to 100) the resulting output more than doubles (from $550 to $1500); in fact it almost triples.

Foolproofing is a process of making an algorithm more reliable, fail-safe, or robust, by anticipating erroneous input or other difficulties. For example, if the hours input are more than the number of hours in a week (7*24 = 168) then an error message should be output. The resulting foolproofed, or robust, algorithm is given in figure 4. It may be further modified to recognize the input of negative hours, and so output another error message. Note that 168 is not shown in the algorithm, for it hides the two "components" of the product. Also, humans shouldn't need to multiply this when computers can do it easily.

Repeating is a very common process with algorithms. For example, the original pay algorithm is shown embedded within a loop in figure 5. When the input values for hours h are greater than or equal to zero, this algorithm computes the pay and keeps repeating. When a negative value is input, the repetition or looping ends.

Embedding an algorithm is the process of re-using that algorithm within another algorithm. Notice that the original algorithm, shown boxed, appears embedded in the various extensions. This shows that when algorithms are structured properly they can be modified without destroying already existing parts. When algorithms are structured poorly, small modifications can cause great problems. Re-Use is also efficient and can save time and effort.

Notice that each of these figures shows a single modification of the original algorithm. If all four modifications were shown on one figure, the resulting algorithm would be much more complex than the original algorithm. It is shown on a following page.

Modifying an already created system is not common in other disciplines. For example, artists do not try to touch up anothers artistic work, and engineers do not add a few extra lanes to an existing bridge; but algorithms are just too easy to modify. The realization that algorithms can be modified throughout their lifetimes is important, for it means that algorithms should be created so as to allow for modification. When algorithms are structured properly their modification can lead to better algorithms; otherwise the modification can be disastrous.