Computer Science 1MD3

Lab 10 – Designing Algorithms

As we know, an algorithm is an ordered set of unambiguous, executable instructions that terminates. However, this really tells us about what algorithms are restricted too, as opposed to how to design them effectively. This lab will outline some simple techniques and practices used in designing algorithms.

PRIMITAVES

In order to eliminate ambiguity in our algorithms it is necessary to define the syntax and corresponding semantics used to represent the algorithm. Syntax are symbols or words that make up a usable language, for instance `if`, `>` and `int` are all in the syntax of the C programming language. The semantics are the implied meanings of all the symbols in the syntax, for instance, > means greater then. Combined, syntax and semantics make up the primitive or your algorithm representation, or programming language.

ABSTRACTION

Abstraction is the notion of hiding detail, when you design a function or procedure you are using abstraction. Abstraction is very useful and pretty much necessary when designing algorithms. Take the example of your hashing assignment. Imagine doing the whole assignment without using the functions `insert()`, or `end()`, it would probably be a lot harder. Abstraction allows us to take for granted operations that seem to be quite trivial on the surface. For instance, how often do we regard multiplication as a function of recursive addition? Not very often.

THE 'ART' OF PROBLEM SOLVING

In your textbook, chapter four deals with the 'art' of problem solving, art however is not really something that can be taught, rather you must to be made to appreciate art. Given this, it must be understood that there are many ways to solve a given problem and that there may be some solutions better than others.

Before you solve a problem it is necessary to understand the question. If you have no idea as to what you are being asked, you certainly cannot offer any answer. The question then must also be unambiguous.

After understanding your question, you must review the tools that you have to solve it. For instance you may need to solve an equation but also have an existing library routine `solveEq`.

The final step in problem solving (when using a computer) is to determine how to take the solution you have in your head and translate it into something that the computer understands.

THE GREEDY ALGORITHM

Typically questions that computer scientists try to solve involve a great deal of computation, like computing Pi or finding paths in graphs. There is a method that theoretically guarantees a solution for these types of questions; the corresponding algorithms are called *greedy algorithms*.

Consider that you are asked to write a program to determine if a positive integer is prime, you could easily take that number and see if it is divisible by any of the numbers below it. Such an algorithm would have to produce a correct result since we are reviewing every possible outcome. This function is informally described below.

isPrime()

```
int isPrime(int x) {

      int count=x;

      for (count=1; count<x; count++) {
            if (x%count==0) then return NOTPRIME;
      }

      return PRIME;

}
```

This algorithm admits correct results but it is terribly inefficient. Analyzing the problem further we realize that the largest divisor of a number, x, is $\frac{x}{2}$, (51 can not possibly go into 100 more then once). Also, something else that may not be that obvious is that we only need to test a given number by dividing it by all the prime numbers before it. This is because any divisor of a number can be broken up into prime factors, so 18 which is divisible by 6 is also divisible by 6's prime factors 2 and 3.

So altering our algorithm we can do this:

isPrime() *better

```
int isPrime(int x) {

      int numprimes=1;
      file primes;

      while(x<=2*readfile(primes)) {
            if (x%readfile(primes)==0) return NOTPRIME;
            advanceInFile(primes);
      }

      return PRIME;

}
```

You may notice that we need a list of primes in order to determine if x is prime, this isn't a problem because we can always use this list to quickly generate a larger prime.

DIVIDE AND CONQUER

Divide and conquer is the process of taking a problem and breaking it into several smaller problems that we can solve. This is often a very efficient way of designing an algorithm.

Suppose we are given an ordered list of number and we would like to determine if a number x is in this list. We can check the middle number of this list (the pivot) and determine if x is larger or smaller then this, taking everything to the left or right of the pivot accordingly. We can continue this process until the pivot equals x or until we are searching an empty list. The corresponding algorithm looks as follows:

Binary Search Algorithm

```
int BSA (list L, int x) {

      if (empty(L)) return FAILURE;
      else {
            if (x==pivot(L)) return SUCCESS;
            else if (x<pivot(L)) {
                  L:=leftOf(pivot(L),L);
                  BSA(L,x);
            }
            else if (x>pivot(L)) {
                  L:=rightOf(pivot(L),L);
                  BSA(L,x);
            }
      }

}
```

Self test questions

1. The puzzle called the Towers of Hanoi consists of three pegs, one of which contains several rings stacked in order of descending diameter from bottom to top. The problem is to move the stack of rings to another peg. You are allowed to move only one ring at a time, and at no time is a ring to be placed on top of a smaller one. Observe that if the puzzle involved only one ring, it would be extremely easy. Moreover, when faced with the problem of moving several rings, if you could move all but the largest ring to another peg, the largest ring could then be placed on the third peg, and then the problem would be to move the remaining rings on top of it. Using the observation, develop a recursive algorithm for solving the Towers of Hanoi puzzle for an arbitrary number of rings.

2. Four prospectors with only one lantern must walk through a mineshaft. At most, two prospectors can travel together and any prospectors in the shaft must be with the lantern. The prospectors, named Andrews, Blake, Johnson, and Kelly, can walk through the shaft in one minute, two minutes, four minutes, and eight minutes, respectively. When two walk together they travel at the speed of the slower prospector. How can all four prospectors get through the mineshaft in only 15 minutes?

3. You are taking a computer science class and would like to do well on the final. What type of method should you employ to guarantee an A.