2005 Paper 1 Question 10

Programming in Java

The naïve way of computing a value to the power n performs $n-1$ multiplications. A much better algorithm, discussed in some detail in the Foundations of Computer Science course, can involve repeated squaring:

```
pow(a, 1) = a
pow(a, n) = pow(a*a, n/2)       [n even]
          = a * pow(a*a, (n-1)/2) [odd]
```

Even this is not always optimal. For example, it computes a^{15} via the sequence $a, a^2, a^3, a^6, a^7, a^{14}, a^{15}$ in 6 steps while it is possible to use $a, a^2, a^4, a^5, a^{10}, a^{15}$ and manage in 5 steps.

Design a Java program that can be given an integer n and will find (by some form of exhaustive search) the smallest number of multiplications that could be used to raise a value to the power n.

The strategy to use is as follows:

Start with the value a before undertaking any multiplication. The first multiplication must therefore be of a by itself, so after one multiplication you have a, a^2 calculated. The second multiplication could multiply a^2 either by itself or by a (there is no point in multiplying a by itself again!) so the situation you reach has either a, a^2, a^4 or a, a^2, a^3 calculated. You will carry on this building up a collection of sets representing the collections of values you might have available after $3, 4, 5, \ldots$ multiplications, and you will use exceptions to exit as soon as you find that one of these collections includes the target exponent n.

The standard Java library class `HashSet` may prove helpful. It supports a method `add` for adding an item, `contains` for checking membership, `equals` for comparing `HashSet`s, and the Java `for (var : collection)` style of iteration can be used to look at all its elements in turn.

It is not expected that your code is fast enough for seriously large values of n.

Minor syntax errors or mistakes about the Java libraries will not be heavily penalised, but code that is without any explanation and that is poorly laid out will not gain many marks.

[20 marks]