CS 302 Computer Fluency
Elaine Rich

Introduction, Bits and Encodings

1. The Brown University Standard Corpus of Present-Day American English, compiled in the 1960s, to support research in computational linguistics, had just over 1,000,000 words. I'm going to estimate an average of 4 characters per word. That means that there were about 4,000,000 characters. At one byte each, the size of the Corpus was about 4 MB. Go to the Project Gutenberg website:

   http://www.gutenberg.org/wiki/Main_Page

   Choose a favorite book. What did you pick? How does the size of that one book (out of the thousands available on just that one website), compare to the size of the Brown Corpus?

2. Memory gets smaller and cheaper every year. One way to see how that has happened over the last decade or so is to look at the history of the iPod. Plot the amount of memory available on the highest end (for purposes of this discussion, the one with the largest memory) model iPod for each year since it came out in 2001. You should be able to get all the data you need at this site, but if some values are missing you can track them down if you know the model names that were available that year:

   http://www.ipodhistory.com/

3. We talked in class about chess. We compared the number of steps required to search a complete game board to the number of seconds since the Big Bang. Pick another two person game. How many branches are there in its search tree? To answer this question, you need to know approximately how many choices there are at each move and how many moves there are in a typical game. It's fine to give your answer in exponent notation.

4. In class, we went through several sets of instructions (recipes, shampoo, etc.) and asked what a robot would have to know in order to implement the instructions effectively. In other words, what the robot would have to know if we want to consider the instructions to be an algorithm. Go find another set of instructions that makes sense to people. Include those instructions as part of your answer to this question. Then list at least five things that an implementing robot would have to know.

5. $10{,}000 = 10^x$ for what value of x?

6. Convert each of the following decimal numbers to binary:
    a. 46
    b. 83
    c. 467

7. Convert each of the following binary numbers to decimal:
    a. 11101
    b. 110111

8. Convert each of the following decimal numbers to hex (Hint: Convert to binary first.):
    a. 159
    b. 234

9. Convert each of the following hex numbers to decimal:
    a. B8
    b. 2E

10. Show the result of each of the following binary arithmetic operations:
    a. 111101 + 111 =
    b. 1101 – 11 =
    c. 10101 * 111 =

11. In this problem, we'll consider what happens when you try to use very large numbers in a Python program. Computers have to make decisions about how to represent numbers. In Python, integers can get arbitrarily large until all of the machine's memory is used. But for nonintegers (think fractions), a different representation (called floating point) is used. A fixed number of bits are allocated to each number. Some of the bits store the significant digits. The rest store the exponent (the power of 10 by which you multiply to get the real number). So, for example, 2.3e2 corresponds to $2.3 * 10^2 = 230$.

    If a number gets too large, there aren't enough bits to represent its exponent. In that case, the machine must do something. Our goal, in designing programs, is to prevent the raising of errors (the things that show up in red). To this end, what is supposed to happen, if the real answer cannot be represented, is that the special value "inf", which stands for infinity, is supposed to be created. When you compute with just the standard "arithmetic" operators, namely +, - , * and /, this will happen. But the Python implementation of exponentiation (raising to a power) happens not to do this. If it generates a number that is too large, it raises the error condition.

    Create a situation where you generate a number that is too large to be represented. We know that this won't happen with integers. So you need to start with a floating point number. You can do this by any use of a decimal point (e.g., 5.2) or by using floating point notation (e.g., 5e2 for 500). Now do operations to make larger and larger numbers until you get either inf or an error. (Go ahead, try things until you find one.) What operations did you perform and what result did you get?

By the way, if you get inf, you might experiment to see what happens if you try to compute with the special "value" inf. If you want to do that, you can grab it using _. What you can't do is just type inf. But you can produce the value any time you want by typing float("inf"). (Go ahead, try things until you find one.)

12. The following string corresponds to the hex description of the ASCII encoding of an English sentence. What is the sentence? (You can find the ASCII equivalence table here: http://www.asciitable.com/ or in my Encodings slides.)

    `4465657020426C756520776F6E20696E20313939372E`

13. Take a look at the Unicode character code chart shown here:

    http://www.unicode.org/charts/

    Pick at least two alphabets you've never heard of. Look to see how they are represented. Show at least two symbols from each (it's fine to draw them as best you can) and indicate their Unicode index number.

14. Describe Texas burnt orange using the hex codes for RGB. There are various applets that you can use to figure this out. Which ones work for you may depend on the security settings on your computer. Here are some you can try. (You may have to click to allow Java to run.)

    http://lectureonline.cl.msu.edu/~mmp/applist/RGBColor/c.htm
    http://www.cbu.edu/~jvarrian/applets/color1/colors_g.htm
    http://easycalculation.com/color-coder.php

15. This is a binary clock:

    Watch this video to see how it works:

    http://www.youtube.com/watch?v=1fFiVjNUjB8. What time is it displaying?