Extra Practice Problems

The final exam will have one question that is longer (and worth more points) than the typical programming questions you've seen on the midterms. This reflects the fact that after working on homework 10, 11 and especially 12, you have developed your skills in writing more complex programs that involve multiple functions.

Here are some "bigger" practice problems to consider. These are challenging but you are strongly encouraged to practice them.

1. You are given a list of dates of birthdays. Each date is one of two formats: year month day as in `'2014/12/31'` or month, day, year as in `'12/31/2014'`. Write a function `build_birthday_dictionary` that takes in such a list and returns a birthday dictionary of a particular form. The keys are month names (you can use three letter abbreviations: "Jan," "Feb," etc.). The values are dictionaries. These inner dictionaries have days for keys and each value is a list of years.

   Example:

```
>>> dates = ['1975/12/17', '1976/12/17',
            '1944/12/11', '1974/03/27']
>>> build_birthday_dictionary(dates)
{'Dec': {'11': ['1944'],
         '17': ['1975', '1976']},
 'Mar': {'27': ['1974']}}
```

   Hint: assume you have a list called `MONTHS` that looks like this:

```
MONTHS = ['Jan', 'Feb', 'Mar', 'Apr', 'May',
          'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
          'Nov', 'Dec']
```

Solution:

```
MONTHS = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov'

def extract_parts(date):
        parts = date.split('/')
        if len(parts[0]) == 4:
                year = parts[0]
                month = parts[1]
                day = parts[2]
        else:
                month = parts[0]
                day = parts[1]
                year = parts[2]
        return [month, day, year]
```

```
def update_bday(m, d, y, bdays):
        if m not in bdays:
                bdays[m] = {}
        if d not in bdays[m]:
                bdays[m][d] = []
        if y not in bdays[m][d]:
                bdays[m][d] += [y]

def build_birthday_dictionary(dates):
        '''(list of str) -> dict
        Expects a list containing

        '''
        bdays = {}
        for date in dates:
                m, d, y = extract_parts(date)

                m = MONTHS[int(m)-1]

                update_bday(m, d, y, bdays)

        return bdays

dates = ['1975/12/17', '1976/12/17',
          '1944/12/11', '1974/03/27']
print build_birthday_dictionary(dates)
```

2. This question has two parts:

   i. Write a function `dna_squish` that takes a sequence of DNA (a string of A, C, G, T) and "compresses" it as follows. Subsequences of the same character are replaced to a single copy of the character followed by the number of times that character occurs. For example, `'ACCCGGCAAAAA'` would be compressed to `'A1 C3 G2 C1 A5'`.

      The resulting string is compressed if the original DNA string contains many repeated characters.

   ii. Write a function `dna_unsquish` that takes as input the string produced by the previous problem and reconstructs the original dna string. On input `'A1 C3 G2 C1 A5'`, it returns `'ACCCGGCAAAAA'`.

   iii. Challenge edition: do not put spaces between the groups when squishing. When unsquishing, make sure your program works even when a group is 10 or larger, e.g., `'C3A10C1'` would produce 'CCCAAAAAAAAAAAC'

Solution:

```
def dna_squish(dna):
        i = 0
        squish = []
        while i < len(dna):
                curr_ch = dna[i]
                count = 0

                while i < len(dna) and dna[i] == curr_ch:
                        count += 1
                        i += 1

                squish += [curr_ch + str(count)]

        return ' '.join(squish)


def dna_squish_v2(dna):
        i = 0
        squish = []
        dna += 'X'  # neat trick: add sentinel to end
                    # makes code a bit simpler
        while dna[i] != 'X':
                curr_ch = dna[i]
                count = 0

                while dna[i] == curr_ch:
                        count += 1
                        i += 1

                squish += [curr_ch + str(count)]

        return ' '.join(squish)


def dna_unsquish(squished):
        dna = ''
        groups = squished.split()
        for group in groups:
                ch = group[0]
                count = int(group[1:])
                dna += ch * count
        return dna

def dna_unsquish_challenge(squished):
        # expects squished string without spaces: like 'A13G2T3G2'
        dna = ''
        i = 0
        while i < len(squished):
                ch = squished[i]
                print 'ch', ch
                i += 1
                count = ''
                while i < len(squished) and squished[i].isdigit():
                        count += squished[i]
                        i += 1
                print 'count', count
                count = int(count)
                dna += ch * count
        return dna

print dna_unsquish_challenge('C3A10C1')
```