Hand-in 6

Python

This is the final hand-in exercise in the course. It is meant to illustrate that it is important to think carefully about how to represent your data. The speed by which a task is solved is very dependent on the way the data is organized. The exercise looks very long, but most of the code that you need to write are minor modifications of earlier hand-ins.

We are going to create a program that compares two fasta-files and calculates how many identical sequences there are in the two files. We will use this program to compare a recent version of swissprot that I've downloaded:

`http://people.binf.ku.dk/wb/data/uniprot_sprot_2013.fasta.gz`

with and old version that we downloaded in 2007:

`http://people.binf.ku.dk/wb/data/uniprot_sprot_2007.fasta.gz`

Note that both files are compressed and need to be unpacked before use.

One way of comparing the files is to put all the protein names from the first file in a list, and thereafter run through a loop where you check for each of these names whether it is present in the second file.

So for each name in file one, we want look up the name in file two to check whether it's present. It is vital that the data in file two is represented in an efficient way. In this exercise we will try three different representations of the data, and thereby three different ways of looking up the names: linear search, binary search and using dictionaries.

When reading in the fasta entries in the files, you probably don't want to store the complete Fasta IDs in your data structure, since they are not necessarily the same in the two files. We recommend that you just extract the ID itself. For instance, for the following fasta label:

```
>11S3_HELAN (P19084) 11S globulin seed storage protein G3 precursor
 (Helianthinin G3) [Contains: 11S globulin seed storage protein G3
 acidic chain; 11S globulin seed storage protein G3 basic chain]
```

You could just store:

```
P19084
```

The format for the labels in FASTA has apparently changed between 2007 and 2013 (this is a frequent problem when working with biological data), making it slightly more complicated to extract this ID. To help you out, here is a helper function which you can use to extract the name from a fasta label:

```
def fasta_label_to_id(label):
```

```
"""Extract meaningful ID from fasta label"""
import re
pattern = re.compile(r'>\w+[ |(]+(\w+)[ |)]')
match = pattern.match(label)
return match.group(1)
```

We have provided two small test files so you can test whether your methods work before actually running on the huge files. These files are called `test1.fasta` and `test2.fasta` and are available in Absalon along with this document.

1. When faced with a new problem, it's always a good idea to consider whether it's really necessary to write your own program for the given task - or whether some existing program will do the job. Last week we used `diff` to compare the contents of two files. Explain why this program is not well suited for this current problem.
2. We want you to create a module file called `handin6.py` with a function called `fasta_to_list` that reads a fasta file (remember to send the file name along as an argument to the function) and returns a list of all protein names (this is a simple version of one of the functions in handin4).
3. Create a test file called `handin6_test1.py` that reads in two files *file1* and *file2* using the `fasta_to_list` function. Now write a loop that for each name in the first list looks through the second list to see if there is a match. It should report the number of names that are in *file1* but not in *file2*. First test `handin6_test1.py` using the test files `test1.fasta` and `test2.fasta`. Then run it on the sprot files using the newest file as *file1* and the older file as *file2*.

   Describe this experience.
4. Create a test file called `handin6_test2.py`, that reads in two files using `fasta_to_list` as before, but now sorts the second list.

   It is much more efficient to find elements in a sorted list. One way of doing this is by using a method called *binary search*. Basically, binary search excludes half of the remaining list at every step of the search and will therefore only look at much fewer elements than the linear search above. To help you out, here is an implementation of a binary search function in Python:

   ```
   def binary_search(list, element):
       '''Search for element in list using binary search.
          Assumes sorted list'''
       # Current active list runs from index_start up to
       # but not including index_end
       index_start = 0
       index_end = len(list)
       while (index_end - index_start)>0:
           index_current = (index_end-index_start)//2 + index_start
           if element == list[index_current]:
               return True
   ```

```
        elif element < list[index_current]:
            index_end = index_current
        elif element > list[index_current]:
            index_start = index_current+1
    return False
```

Write a loop that for each element in the first list uses the provided binary search function to find the element in the second list. Like before, use `handin6_test2.py` on the test files to convince yourself that it works. When convinced, run it on the sprot files. Does this improve the performance?

5. Finally, add a function called `fasta_to_dict` to `handin6.py` that reads a fasta file and returns a dictionary containing the names as keys (a simple version of a function that you wrote for handin4). Create a test file called `handin6_test3.py` where you read in the first file as a list using `fasta_to_list` and read the second file in as a dictionary using `fasta_to_dict`. Now write a loop that for each element in the list looks the name up in the dictionary, again reporting the number of names that are present in the first file but not in the second. Test it on the test files to see if it works. Then test it on the sprot files. Describe the performance.

Unix

Use the Unix `time` command to quantify the difference in execution time for the different solutions above.

Please hand in the files `handin6.py`, `handin6_test1.py`, `handin6_test2.py` and `handin6_test3.py`, and the output of the different tests.

Please note that the `fasta_label_to_name` and `binary_search` functions have been made available as code on the Absalon page (the `handin6.py` file) to avoid problems with copy & pasting from this PDF document.

As usual, don't hesitate to contact us if you're stuck on some problem.