Week 10 - Lecture 19

István Albert
BMB, Bioinformatics
Expand our previous program

• We now know how to find unique starts/ends

• Which sequences cause the most non-uniqueness to occur?

• For each “start” store the sequences that contain that start

  \[ \text{start} \rightarrow \text{key} \]
  \[ \text{sequences} \rightarrow \text{value} \]
from Bio import SeqIO

stream = open('ecoli.gb')
records = list(SeqIO.parse(stream, 'gb'))

def extract(rec):
    id = rec.id
    start = str(rec.seq[:5])
    return start, id

recs = map(extract, records)
print recs[0:3]

python -t C:\cygwin\home\albert\docs\albert-web\ppt\fall-2010\597D\week10\work.py 2>&1` returned 0

[('GCGAT', 'AB028471.1'), ('AACAA', 'AY249138.1'), ('ATTAC', 'AB128919.1')]
This produces unique keys, but won’t keep all values

```python
recs = map(extract, records)
store = dict(recs)
print store['ATGGC']
print len(store.keys())
print store.keys()
```

Command Output:

```
AY250029.1
10
['ATGGC', 'CCAGT', 'GGGAT', 'AACAA', 'ATTAC', 'TGAGT', 'GCGCT', 'ACTTG', 'GCGAT', 'ACTCG']
```
Create a key ➔ list mapping

```python
# populate dictionary with lists as values
store = dict()
for id, seq in recs:
    store[id] = []

print store['ATGGC']
print len(store.keys())
print store.keys()
```

```
[ ]
10
['ATGGC', 'CCAGT', 'GGGAT', 'AACAA', 'ATTAC', 'TGAGT', 'GCGCT', 'ACTTG', 'GCGAT', 'ACTCG']
```
Iterate twice once to create, then to populate

```python
# populate dictionary with lists as values
store = dict()
for id, seq in recs:
    store[id] = []

# now you can append to each value
for id, seq in recs:
    store[id].append(seq)

print store['ATGGC']
```

Output:
```
[ 'AY249989.1', 'AY249991.1', 'AY249994.1', 'AY249995.1', 'AY249998.1', 'AY249999.1', 'AY250001.1', 'AY250002.1', 'AY250003.1', 'AY250005.1', 'AY250008.1', 'AY250010.1'
```
for id, seq in recs:
    store[id].append(seq)

# count the matches for each start
for key, value in store.items():
    print key, len(value)
Default dictionaries – skip intialization

```python
# default dictionaries allow you to skip initialization
from collections import defaultdict
store = defaultdict(list)

# you can directly append to each value
for id, seq in recs:
    store[id].append(seq)

print store['ATTAC']
```

```
['AB128919.1', 'AB128920.1', 'AB128921.1']
```
Pattern Matching with strings

Strings only have very simple methods. These can be very useful but are limited.

```python
def fix(text):
    "Fix address by replacing ROAD with RD."
    return text.replace('ROAD', 'RD.')

addr1 = '100 NORTH MAIN ROAD'
addr2 = '100 NORTH BROAD ROAD'
addr3 = '100 NORTH BROAD RD'

print fix(addr1)
```

Output:
```
100 NORTH MAIN RD.
```
Pattern matching with regular expressions

```python
import re

daddr1 = '100 NORTH BROAD ROAD'

# re allows us to substitute special characters
# to indicate certain patterns
# $ indicates end of line
patt = "ROAD$"
repl = "RD."

print re.sub(patt, repl, addr1)
```

```
100 NORTH BROAD RD.
```
Special characters

- `'.'` (Dot.) In the default mode, this matches any character except a newline.
- `'^'` (Caret.) Matches the start of the string
- `'\$'` Matches the end of the string or just before the newline at the end of the string
- `'\*'` Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible
- `'+'` Causes the resulting RE to match 1 or more repetitions of the preceding RE
- `'?'` Causes the resulting RE to match 0 or 1 repetitions of the preceding RE.

And many others: [http://docs.python.org/library/re.html](http://docs.python.org/library/re.html)
More special characters

- \w matches any alphanumerical character and the underscore
- \W matches any non-alphanumerical character
- \s matches whitespace
- \S matches non whitespace
- | matches one or the other pattern
Explore the pattern matching

- Enter a Perl-style regular expression:
  - Mary
- Pattern options:
  - IGNORECASE
  - LOCATE
  - MULTILINE
  - DOTALL
  - VERBOSE
- Enter a string to search:
  - Mary had a little lamb! Hello Mary!
- Highlight options:
  - Highlight first match
  - Highlight all matches
- Groups:
  - 0: u'Mary'
```python
def extract(rec):
    id = rec.id
    seq = str(rec.seq)
    return id, seq

recs = map(extract, records)

patt = "ATTT"
for id, seq in recs:
    found = re.findall(patt, seq)
    print id, len(found)
```
What can you do with re

• Find matches
• Substitute values
• Split strings

Then for each you can find where it matches, find groups, turn case sensitivity on and off etc

Google ➔ Python: Regular Expression Howto
http://docs.python.org/howto/regex.html
Homework 19

• Find the sequence ids that cause the most common (start, end) pairs when sliced for 10 elements

• Create a pattern that matches the following:

A followed by C or G then one or more T or A
Sorting

• A fundamental process necessary for solving just about any computational problem

• Searching is simple if the elements are already ordered

**Note:** if you need to search for something, see if sorting would solve the problem.
<table>
<thead>
<tr>
<th>Name</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
<th>Memory</th>
<th>Stable</th>
<th>Method</th>
<th>Other notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timsort</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion &amp; Merging</td>
<td>$n$ comparisons when the data is already sorted.</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td>Tiny code</td>
</tr>
<tr>
<td>Cocktail sort</td>
<td>—</td>
<td>—</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td></td>
</tr>
<tr>
<td>Comb sort</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1</td>
<td>No</td>
<td>Exchanging</td>
<td>Small code size</td>
</tr>
<tr>
<td>Gnome sort</td>
<td>—</td>
<td>—</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td>Tiny code size</td>
</tr>
<tr>
<td>Selection sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Insertion sort</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Insertion</td>
<td>Average case is also $\mathcal{O}(n + d)$, where $d$ is the number of inversions</td>
</tr>
<tr>
<td>Cycle sort</td>
<td>—</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Insertion</td>
<td>In-place with theoretically optimal number of writes</td>
</tr>
<tr>
<td>Shell sort</td>
<td>—</td>
<td>—</td>
<td>$n \log^2 n$</td>
<td>1</td>
<td>No</td>
<td>Insertion</td>
<td></td>
</tr>
<tr>
<td>Binary tree sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion</td>
<td>When using a self-balancing binary search tree</td>
</tr>
<tr>
<td>Library sort</td>
<td>$n \log n$</td>
<td>$n^2$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion</td>
<td></td>
</tr>
<tr>
<td>Merge sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>Depends</td>
<td>Yes</td>
<td>Merging</td>
<td>Used to sort this table in Firefox [3].</td>
</tr>
<tr>
<td>In-place merge sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>Depends</td>
<td>Yes</td>
<td>Merging</td>
<td>Example implementation here: [4]. can be implemented as a stable sort based on stable in-place merging: [5].</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>Depends</td>
<td>Merging</td>
<td></td>
</tr>
<tr>
<td>Smoothsort</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td>An adaptive sort - $n$ comparisons when the data is already sorted, and $\mathcal{O}(1)$ swaps.</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n^2$</td>
<td>$\log n$</td>
<td>Depends</td>
<td>Partitioning</td>
<td>Can be implemented as a stable sort depending on how the pivot is handled. Native variants use $\mathcal{O}(n)$ space</td>
</tr>
<tr>
<td>Intrasort</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>No</td>
<td>Hybrid</td>
<td>Used in SGI STL implementations</td>
</tr>
<tr>
<td>Patience sorting</td>
<td>—</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>No</td>
<td>Insertion &amp; Selection</td>
<td>Finds all the longest increasing subsequences within $O(n \log n)$</td>
</tr>
<tr>
<td>Strand sort</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n^2$</td>
<td>$n$</td>
<td>Yes</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Tournament sort</td>
<td>—</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td></td>
<td>Selection</td>
<td></td>
</tr>
</tbody>
</table>
Default “tuple” sort

```python
a = [(1, 'C'), (1, 'A'), (3, 'A'), (2, 'B')]
b = sorted(a)
print(b)
```

It is in the correct order by field 1 then by field 2!
Works for any number of fields

```python
a = [(1,2,1), (1,1,3), (2,3,1), (3,2,1),
     (3,1,2), (2,3,2)]

b = sorted(a)

print b
```

```
[(1, 1, 3), (1, 2, 1), (2, 3, 1), (2, 3, 2), (3, 1, 2), (3, 2, 1)]
```
Decorate ➔ Sort ➔ Un-decorate
DSU pattern

Decorate → Sort → Un-decorate

• To sort an a series of objects by certain fields all we need is to create an extended (decorated) version of the data that contains fields that we want to sort by
Decorate-Sort example

```python
# sort this list by increasing name length

data = [ 'Jennifer', 'Janet', 'Jasmine', 'Jill' ]
```

```
python -tu C:\cygwin\home\ialbert\sources\ialbert-web\ppt\week6\lecture12\demo.py 2>&1` returned 0
```
def decorate(name):
    return len(name), name

names = [ 'Jennifer', 'Janet', 'Jasmine', 'Jill' ]
decor = map(decorate, names)
print decor

decor = sorted(decor)
print decor
Un-decorate

```python
print decor

decor = sorted(decor)
print decor

def undecorate(elems):
    return elems[-1]

print map(undecorate, decor)
```

```
[(8, 'Jennifer'), (5, 'Janet'), (7, 'Jasmine'), (4, 'Jill')]
[(4, 'Jill'), (5, 'Janet'), (7, 'Jasmine'), (8, 'Jennifer')]
['Jill', 'Janet', 'Jasmine', 'Jennifer']
```
from Bio import SeqIO

stream = open( 'ecoli.gb' )
recs = list( SeqIO.parse(stream, 'gb') )

print recs[0]
```python
stream = open('ecoli.gb')
recs = list(SeqIO.parse(stream, 'gb'))

def decorate(rec):
    return len(rec.seq), rec

decs = map(decorate, recs)

print decs[0]
```

```python
(2351,
 SeqRecord(seq=Seq('GCGATTTTCCTTTTATCTTTGCACACGTAAAACGAATA
CCGGGGTTATCGGCCTG...TCA', IUPACAmbiguousDNA()),
    id='AB028471.1', name='AB028471')
```
Sort – descending order

```python
recs = list(SeqIO.parse(stream, 'gb'))

def decorate(rec):
    return len(rec.seq), rec

decs = map(decorate, recs)
decs = sorted(decs, reverse=True)

print decs[0]
```

```
(12193,
 SeqRecord(seq=Seq('ACTCGTCTCTCCGTCTGCATTTTTTAGGTGTATAACGC
 CCAAATATTCAACGAT...', CGA'), IUPACAmbiguousDNA()),
 id='EF392693.1', name='EF392693'),
```
Recall the pattern matching

```python
import re

text = "AATAAGCCGAGAAG"
patt = "AAT|AAG"

print re.findall(patt, text)
matches = re.finditer(patt, text)
for m in matches:
    print m.start(), m.end()
```
Student data is stored as tuples representing name, gender and age:

\[
[ ('John', 'M', 22), ('Jack', 'M', 19), ('Jill', 'F', 21) ]
\]

Sort this data by gender, age and name to obtain:

\[
[ ('Jill', 'F', 21), ('Jack', 'M', 19), ('John', 'M', 22) ]
\]
Homework 20 (part 2)

Sort the e-coli strains in decreasing order by:

1. The times they contain the pattern AAAA or TTTT, then their length, then their ID, here is an output that you might get for the first 10 elements:

```
[(167, 12193, 'EF392693.1'), (106, 9374, 'AB269770.1'), (93, 7987, 'AB269771.1'), (31, 2676, 'AB128920.1'), (27, 2351, 'AB028471.1'), (27, 2329, 'AB028474.1'), (26, 1996, 'AB128919.1'), (24, 2359, 'AB028475.1'), (20, 1506, 'AY250000.1'), (19, 1740, 'AY250010.1')]
```