Searching and Regular Expressions

Proteins

- 20 amino acids
- Interesting structures
  - beta barrel, greek key motif, EF hand ...
- Bind, move, catalyze, recognize, block, ...
- Many post-translational modifications
- Structure/function strongly influenced by sequence

Sequence Suggests Structure/Function

When working with tumors you find the p53 tumor antigen, which is found in increased amounts in transformed cells.

After looking at many p53s you find that the substring `MCNSSCMGGMNRR` is well conserved and has few false (mis)matches.

If you have a new protein sequence and it has this substring then it is likely to be a p53 tumor antigen.

Finding a string

We've covered several ways to find a substring in a larger string.

`site in sequence` -- test if the substring *site* is found anywhere in the *sequence*

`sequence.find(site)` -- find the index of the first *site* in the *sequence*. Return -1 if not found.

`sequence.count(site)` -- count the number of times *site* is found in the *sequence* (no overlaps).

Is it a p53 sequence?

```
>>> p53 = "MCNSSCMGGMNRR"
>>> protein = "SEFTTVLYNFMCNSSCMGGMNRRPILTIIS"
>>> protein.find(p53)
10
>>> protein[10:10+len(p53)]
'MCNSSCMGGMNRR'
>>>
```

p53 needs more than one test substring

After a while you find that p53s are variable in one residue.

```
MCNSSCMGGMNRR
```

or

```
MCNSSCVGGMNRR
```

You could test for both cases, but as you add more possibilities the number of patterns gets really large, and writing them out is tedious.

Need a *pattern*

Rather than write each alternative, perhaps we can write a pattern, which is used to describe all the strings to test.

MCNSSC**M**GGMNRR
or
MCNSSC**V**GGMNRR

$\Rightarrow$

MCNSSC**[MV]**GGMNRR

Use [] to indicate a list of residues that could match.

`[FILAPVM]` matches any hydrophobic residue

PROSITE

PROSITE is a database of protein patterns.

http://au.expasy.org/prosite/

The documentation for a pattern is in PRODOC.

PROSITE contains links to SWISS-PROT (a protein sequence database) and PDB (a structure database)

ANTENNAPEDIA

'Homeobox' antennapedia-type protein signature.

Look for a substring which:

[LIVMFE][FY]PWM[KRQTA]

$\swarrow$

Starts with L, I, V, M, F, or E

ANTENNAPEDIA

'Homeobox' antennapedia-type protein signature.

Look for a substring which:

[LIVMFE][FY]PWM[KRQTA]

Starts with L, I, V, M, F, or E

Then has an F or Y

ANTENNAPEDIA

'Homeobox' antennapedia-type protein signature.

Look for a substring which:

[LIVMFE][FY]PWM[KRQTA]

Starts with L, I, V, M, F, or E

Then has an F or Y

Then the letter P

Followed by a W

Followed by an M

ANTENNAPEDIA

'Homeobox' antennapedia-type protein signature.

Look for a substring which:

[LIVMFE][FY]PWM[KRQTA]

Starts with L, I, V, M, F, or E

Then has an F or Y

Then the letter P

Followed by a W

Followed by an M

And ending with a K, R, Q, T, or A

Find ANTENNAPEDIA

Can you find [LIVMFE][FY]PWM[KRQTA] ?

```
MDPDCFAMSS YQFVNSLASC YPQQMNPQQN HPGAGNSSAG GSGGGAGGSG GVVPSGGTNG
GQGSAGAATP GANDYFPAAA AYTPNLYPNT PQPTTPIRRL ADREIRIWWT TRSCSRSDCS
CSSSSNSNSS NMPMQRQSCC QQQQQLAQQQ HPQQQQQQQQ ANISCKYAND PVTPGGSGGG
GVSGSNNNNN SANSNNNNSQ SLASPQDLST RDISPKLSPS SVVESVARSL NKGVLGGSLA
AAAAAAGLNN NHSGSGVSGG PGNVNVPMHS PGGGDSDSES DSGNEAGSSQ NSGNGKKNPP
QIYPWMKRVH LGTSTVNANG ETKRQRTSYT RYQTLELEKE FHFNRYLTRR RRIEIAHALC
LTERQIKIWF QNRRMKWKKE HKMASMNIVP YHMGPYGHPY HQFDIHPSQF AHLSA
```

That's why we have computers.

Sequences with the ANTENNAPEDIA motif

Here are some sequences which contain substrings which fit the pattern

[LIVMFE][FY]PWM[KRQTA]

...LHNEANLR**IYPWMR**SAGADR...

...PTVGKQ**IFPWMK**ES...

...**VFPWMKMGGAKGGESKRTR**...

Not a given residue

Suppose you know from structural reasons that a residue cannot be a proline. You could write

[ACDEFGHIKLMNQRSTVWY]

That's tedious, so let's use a new notation

[^P]

This matches anything which is *not* a proline.
(Yes, using the ^ is strange. That's the way it is.)

N-glycosylation site

This is the pattern for PS00001, ASN_GLYCOSYLATION

```
N[^P][ST][^P]
```

Match an N,
Then anything which isn't a P,
Then an S or T,
And finally, anything which isn't a P

Allow anything

Sometimes the pattern can have anything in a given position - it just needs the proper spacing.

Could use [ACDEFGHJKLMNPQRSTVWY] but that gets tedious. (Have you noticed how often I use that word?) Instead, let's make a new notation for "anything"

Let's use the dot, ".", so that `P.P` matches a proline followed by any residue followed by a proline.

Barwin domain signature 1

The pattern is: `CG[KR]CL.V.N`

The substring must start with a C,
second letter must be a G,
third must be a K or R,
fourth must be a C,
fifth must be an L,
sixth may be any residue,
seventh must be a V,
eight may also be any residue,
last must be an N.

...SSCGKCLSVTNTG...

Repeats

Sometimes you'll repeat yourself repeat yourself. For example, a pattern may require 5 hydrophobic residues between two well conserved regions.

You could write it as

```
[FILAPVM][FILAPVM][FILAPVM][FILAPVM][FILAPVM]
```

but that gets tedious. Again that word. And again we'll create a new notation. Let's use {}s with a number inside to indicate how many times to repeat the previous pattern.

[FILAPVM]{5}

[FILAPVM]{5}

The {}s repeat the previous *pattern*.
The above matches all of the following

```
AAAAA
AAPAP
LAPMAVAILA
VILLAMAP
LAPLAMP
```

And `.{6}` matches any string of at least length 6.

EGF-like domain signature 1

The pattern for PS00022 is: `C.C.{5}G.{2}C`

Match a C, followed by any residue, followed by a C, followed by 5 residues of any type, then a G, then 2 of any residue type, then a C.

...`VCSNEGKCICQPDWTGKDCS`...

Count Ranges

Sometimes you may have a range of repeats. For example, a loop can have 3 to 5 residues in it. All of our patterns so far only matched a fixed number of characters, so we need to modify the notation.

{m,n} - repeat the previous pattern at least m times and up to n times.

For example, A{3, 5} matches AAA, AAAA, and AAAAA but does not match AA nor AATAA.

EGF-like domain signature 2

PS01186 is: `C.C.{2}[GP][FYW].{4,8}C`

Use a spacer of at least 4 residues and up to (and including) 8 residues.

```
RHCYCEEGWAPPDCTTQLKA
RHCYCEEGWAPPDECTTQLKA
RHCYCEEGWAPPDEQCTTQLKA
RHCYCEEGWAPPDEQWCTTQLKA
RHCYCEEGWAPPDEQWICTTQLKA
```

Short-hand versions of counts ranges

This notation is very powerful and widely used outside of bioinformatics. (I think research on it started in the 1950s). Some repeat ranges are used so frequently that (to prevent tedium, and to make things easier to read) there is special notation for them.

			What it means
`{0, 1}`	$\longrightarrow$	`?`	"optional"
`{0,}`	$\longrightarrow$	`*`	"0 or more"
`{1,}`	$\longrightarrow$	`+`	"at least one"

N- and C- terminals

Some things only happen at the N- terminal (start of the sequence) or C-terminal (end of the sequence). We don't have a way to say that so we need - yes, you guessed it - more notation.

^ means the start of the sequence (a ^ inside of [ ] s means "not", outside means "start")

$ means ends of the sequence

^examples$

^A	start with an A
^[MPK]	start with an M, P, or K
E$	end with an E
[QSN]$	end with a Q, S, or N
^[^P]	start with anything except P
^A.*E$	start with an A and end with an E

Neuromodulin (GAP-43) signature 1

The pattern for PS00412 is: `^MLCC[LIVM]RR`

Does match: `MLCCIRRTKPVEKNEEADQE`

Does not match: `MMLCCIRRTKPVEKNEEADQE`

Endoplasmic reticulum targeting sequence

The pattern for PS00014 is: `[KRHQSA][DENQ]EL$`

Does match: `ADGGVDDDHDEL`

Does not match: `ADGGVDDDHDELQ`

Regular expressions

These sorts of patterns which match strings are called "regular expressions". (The name "regular" comes from a theoretical model of how simple computers work, and "expressions" because they are written as text.)

People don't like saying "regular expression" all the time so will often say "regexp", "regex", or "re", or (rarely) "rx".

Many different regexp languages

We've learned a bit of the "perl5" regular expression language. It's the most common and is used by Python and other languages. There's even pcre (*perl compatible regular expressions*) for C.

There are many others: grep, emacs, awk, POSIX, and the shells all use different ways to write the same pattern.

PROSITE also has its own unique form (which I didn't teach because no one else uses it).

regexps in Python

The re module in Python has functions for working with regular expressions.

```
>>> import re
>>>
```

The ‘search’ method

```
>>> import re
>>> text = "My name is Andrew"
>>> re.search(r"[AT]", text)
```

The first parameter is the pattern, as a string.
The second is the string to search.

I use a r“raw” string here. Not needed, but you should use it for all patterns.

The Match object

```
>>> import re
>>> text = "My name is Andrew"
>>> re.search(r"[AT]", text)
<_sre.SRE_Match object at 0x3f8d40>
```

The search returns a "Match" object. Just like a file object, there is no simple way to show it.

Using the match

```
>>> import re
>>> text = "My name is Andrew"
>>> re.search(r"[AT]", text)
<_sre.SRE_Match object at 0x3f8d40>
>>> match = re.search(r"[AT]", text)
>>> match.start()
11
>>> match.end()
12
>>> text[11:12]
'A'
>>>
```

Match a protein motif

```
>>> pattern = r"[LIVMFE][FY]PWM[KRQTA]"
>>> seq = "LHNEANLRIYPWMRSAGADR"
>>> match = re.search(pattern, seq)
>>> match.start()
8
>>> match.end()
14
>>>
```

If it doesn’t match..

The search returns nothing (the None object) when no match was found.

```
>>> import re
>>> pattern = r"[LIVMFE][FY]PWM[KRQTA]"
>>> match = re.search(pattern,
"AAAAAAAAAAAAAA")
>>> print match
None
>>>
```

List matching patterns

```
>>> import re
>>> pattern = r"[LIVMFE][FY]PWM[KRQTA]"
>>> sequences = ["LHNEANLRIYPWMRSAGADR",
...              "PTVGKQIFPWMKES",
...              "NEANLKQIFPGAATR",
...              "VFPWMKMGGAKGGESKRTR"]
>>> for seq in sequences:
...   match = re.search(pattern, seq)
...   if match:
...     print seq, "matches"
...   else:
...     print seq, "does not have the motif"
...
LHNEANLRIYPWMRSAGADR matches
PTVGKQIFPWMKES matches
NEANLKQIFPGAATR does not have the motif
VFPWMKMGGAKGGESKRTR matches
>>>
```

Groups

Suppose an enzyme modifies a protein, and recognizes the portion of the sequence matching
[ASD]{3,5}[LI][^P]{2,5}
The modification only occurs on the [IL] residue. I want to know the residue of that one residue, and not the start/end positions of the whole motif. This requires a new notation, groups.

(groups)

Use ()s to indicate groups. The first (is the start of the first group, the second (is the start of the second group, etc. A group ends with the matching).

```
>>> import re
>>> pattern = r"[ASD]{3,5}([LI])[^P]{2,5}"
>>> seq = "EASALWTRD"
>>> match = re.search(pattern, seq)
>>> print match.start(), match.end()
1 9
>>> match.start(1), match.end(1)
4 5
>>>
```

Parsing with regexps

Groups are great for parsing. Suppose I have the string

```
Name: Andrew  Age: 33
```

and want to get the name and the age values. I can use a pattern with a group for each field.

```
Name: ([^ ]+) +Age: ([0123456789]+)
```

Dissecting that pattern

```
Name: ([^ ]+) +Age: ([0123456789]+)
```

Start with "Name: "

"Age: "

One or more non-space characters (group 1)

One or more digits (group 2)

One or more spaces

Shorthand

Saying [0123456789] is tedious (again!)
There is special shorthand notation for some of the more common sets.

```
Name: ([^ ]+) +Age: (\d+)
```

Some others

`\d` = [0123456789]
\w = letters, digits, and the underscore
\s = “whitespace” (space, newline, tab, and a few others)

Using it

```
>>> import re
>>> text = "Name: Andrew Age: 33"
>>> pattern = r"Name: ([^ ]+) +Age: ([0123456789]+)"
>>> match = re.search(pattern, text)
>>> match.start(1)
6
>>> match.end(1)
12
>>> match.group(1)
'Andrew'
>>> match.group(2)
'33'
>>>
```