課題 2-2,2-3（課題 2-1 発展編）

0.1 課題 2-2,2-3

・2-2：前章で作ったプログラムを改良し、担当遺伝子配列のアミノ酸配列を 3 通りすべて求める。

・2-3：相補的な DNA 配列を求めるプログラムを書く。担当遺伝子の相補鎖の DNA 配列を求め、そのアミノ酸配列を 3 通り求める。2-2 の結果と合わせて 6 通りの Reading Frame のうち、最も確からしい遺伝子領域はどれかを考察する。

1 課題 2-2

みなさん先週の翻訳プログラムはうまくできたでしょうか。先週は配列ファイルの先頭から 3 つずつを順々にアミノ酸に翻訳していきました。

が、よく考えると、2 番目の塩基から翻訳した場合、3 番目の塩基から翻訳した場合では結果が異なってしまいます。

```
C A T G C T G A C
His___Ala___Asp___
  Met___Leu___Thr_____
    Cys   Stop
```

図のなかでは、第 1 のケースではこれは単なる配列です。しかし第 2 のケースではメチオニンが現れていますから、もしかしたらこれが遺伝子のはじまりかもしれません。さらに第 3 のケースではストップコドンが現れています。これが遺伝子の途中だとしたら、ここでコード領域が終わることになります。みなさんが切り取ったファイルはマイコプラズマ菌のゲノムを何の根拠もなくぶった切っただけですから、その先頭から 3 つずつ読んでいってきちんと遺伝子に探し当たる証拠はありません。

そこで、この 3 つのケースすべてを試してみる必要があります。

方法は、2 つあります。

・先頭の塩基をいくつか削って残りの 2 通りファイルをつくり、前回つくったプログラムにかける

・3 通りのケースすべてをプログラムで処理する

どちらの方法でもよいのですが、どうせですからここではプログラムを書くことにします。1 回書いてしまえばあとは実行するだけですから楽です。

1.1 前回のプログラムをもとに翻訳

前回つくったなかで、実際に 3 文字ずつ翻訳するサブルーチンを作ったのですから、これを改良

して塩基配列の 1 文字目からと、2 文字目からと、3 文字目からに対して実行すればいいわけです。

つまり、for 文の開始がいままでは $i = 0 になっていたと思いますが、この初期値を $j などにしてやり、もう一つ外側に for 文を作り、$j が 0 から 2 までを繰り返してやれば 3 つの読み枠に対応できます。この時、返すアミノ酸配列も 3 つになるのですから、せっかくなのでこの $j を使って @amino という配列に記憶させてみましょう。こうしておくと、最後の return で配列を返せばいいので楽です。

2 課題 2-3

課題 2-2 で reading frame の 3 つの可能性をすべて表示することができるようになりました。しかし、実は可能性はまだあります。それは、DNA の二重螺旋の反対側にある場合です。

```
5' __G T A C G A C T G __3'
3'   C A T G C T G A C   5'
```

DNA 鎖は分子内のリン酸の結合位置によって 5' 末端と 3' 末端の方向性があります。遺伝子は、その発現機構の性質で、5' 側から 3' 側へと向かう方向にしかありません。

DNA の二重螺旋は 2 つの逆の方向を持った鎖によって構成され、互いに相補的な A と T、G と C によって互いに結合しています。

というわけで、いまある片方の配列からもう片方の配列を得るには、

1. いまある配列を逆向きにする

2. それから塩基をすべて相補的な塩基に置き換える

という操作をしなければなりません。こうして得られた配列の 3 通りを加えて合計 6 通りの読み方を考慮に入れれば、完璧です。

2.1 相補鎖を求める関数

まず配列をひっくり返さなければなりませんが、これにはせっかくなので新しい関数を作ってしまいましょう。$seq を引数で渡すと、その相補鎖を返してくれる関数ができれば便利です。このために必要な手順は、配列の順序を入れ替えることと、A ならば T、G ならば C といった具合に相補塩基を当てはめることです。

まず、関数の枠組み自体は前回同様に作ります。

```
sub complemental(){
        my $nuc = shift;
        my $complement = '';

        ************** # 裏返す
        ************** # 相補塩基を当てはめる

        return $complement;
}
```

こんな時、文字列の順序を入れ替えてくれる関数があったら便利ですよね。しかし、そんなうまい話が、、、あるんです。Perl には標準で reverse という関数があり、$a = reverse("hoge"); としてやると $a に "egoh" が入ります。Larry Wall に感謝ですね。

さて、さらに相補塩基を当てはめる関数なんかがあればうれしいのですが、それはまだ G-language や BioPerl などが必要で、標準では存在しません。しかし、Perl はもちろんこれを簡単にする仕組みを用意してくれています。そう、勘のいい人は気が付いたでしょう。tr/// です。

tr/// は一文字置換の関数なのですが、これは非常に便利で、特定の文字がきたら特定の文字に、という複数の条件を一文で書き表すことができます。つまり、

```
$nuc =~ tr [atgc]
           [tacg];
```

と書くだけで、a を t に、g を c に置換してくれるのです。Perl はこのように非常に気の利いた言語ですので、とことん楽をしようと考えれば考えるほど、簡単に、綺麗なプログラムが書ける可能性が高くなると言っても過言ではないでしょう。

それでは、プログラムを書いてみましょう。できれば前回学んだように、できるだけ短くまとめてみましょう。

３　遺伝子（コード領域）を見つける

これでみなさんの担当の塩基配列について、全ての可能性でのアミノ酸翻訳結果が出揃ったことになります。では、生体内で実際にアミノ酸に翻訳される部分はどうやって見つければいいでしょうか。

厳密には、これはウエットな実験によって決定されるべきものですが、塩基配列の字面からだけでもある程度有効な予測をすることが可能です。

そのルールは以下のようなものです。

1. コード領域は atg（M, メチオニン）からはじまる。

2. バクテリアの場合はまれに gtg からはじまる。これは本来バリンのコードだが、スタートコドンとなる場合に限ってメチオニンに翻訳される。

前回、これを V と表示するようにプログラムしたはずです

3. コード領域は taa,tga,tag いずれかのストップコドンが終端となる。

4. 複数の可能性がある場合、1 番長いものである可能性が高い。

ほとんどの場合、このルールに沿って探していけば、これだ！というものが見つかるはずです。

これだけでは判断が難しい場合は、ケースバイケースでしょう。

これをプログラムで自動化してみてもいいですね。興味と余力のある人はやってみるとよいでしょう。

４　　おまけ・より高度なプログラミングのために

遺伝子情報処理の扱う範囲はあくまで Biology です。コンピュータをこの分野に使うことは決して単なる手段ではなく、コンピュータを使うことこそが本質的な事柄となり得るのですが、しかし実際の研究ではプログラムを書くこと自体は副次的な作業です。

とすれば、だからこそ、不必要に研究期間を延長することなく、また相互にプログラム資源を共有するために、読み易く、1 回限りの使用で終わってしまわないプログラムを書くことが重要となります。

なるべく汎用性を意識し、繰り返す処理は関数化していくように心がけると良いでしょう。そして、ある程度 Perl に慣れてきたら、「Effective Perl」を読むことを薦めます。きっと数倍効率よく Perl をかけるようになるはずです。

それでは、これからの研究を頑張ってください。