

# ゲノムデータベースとゲノムブラウザ

法政大学 生命科学部 大島研郎

## 本日の講義資料

- kiso2 本日の講義で使用するWebページへのリンクが載せてあります。
- blast.pl
- result.txt

本日の講義では、Active perl を使います。

◆ コマンドプロンプトを立ち上げてください

スタート → すべてのプログラム → アクセサリ → コマンドプロンプト

```
> perl -v
```

と入力して、エラーが出ないことを確認してください

# ゲノムとは

gene(遺伝子) + -ome(総体)

ゲノム = ある生物のもつ全ての遺伝情報

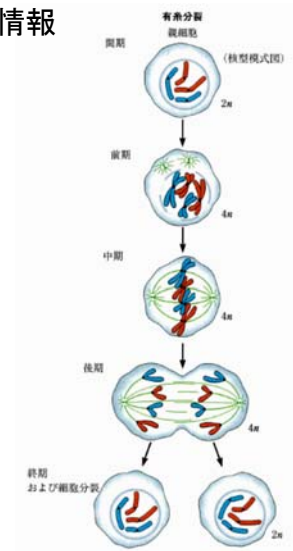
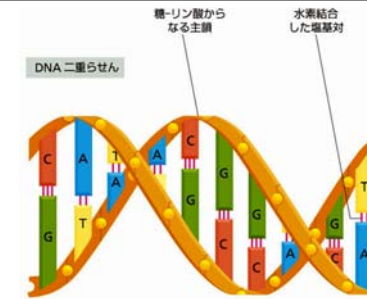
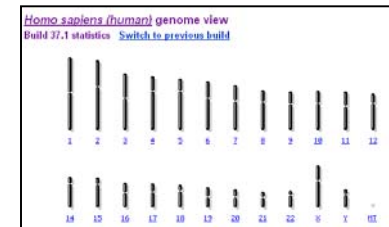
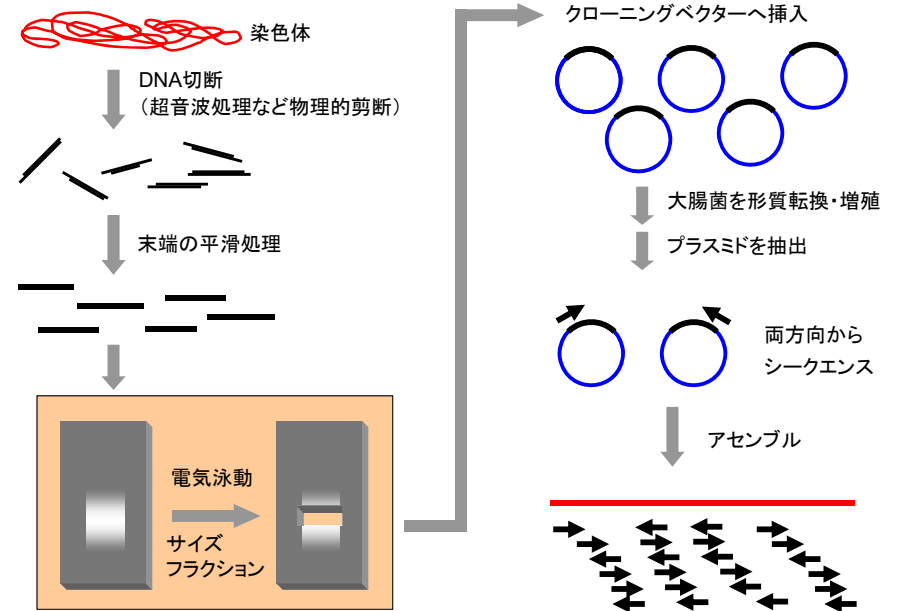


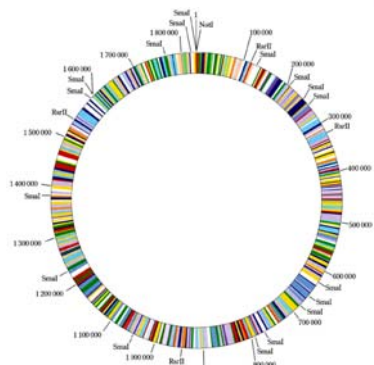
表 1-1 配列が完全に決定されたゲノムの例

生物種	特徴	生息場所	ゲノムサイズ (一倍体ゲノムあたりの塩基対数、×1000)	タンパク質指令遺伝子の数(推定)
<b>細菌</b>				
マイコプラズマの一種 <i>Mycoplasma genitalium</i>	既知の細胞ゲノムのうちで最小のゲノムをもつ	ヒトの生殖道	580	468
<i>Synechocystis</i> sp.	光合成を行い、酸素を作り出す(シアノバクテリアの一種)	湖や小川	3573	3168
大腸菌 <i>Escherichia coli</i>	実験室でよく使われる	ヒトの腸	4639	4289
ヘリコバクター・ピロリ <i>Helicobacter pylori</i>	胃潰瘍を起こし、胃がんの原因となる	ヒトの胃	1667	1590
<b>真核生物</b>				
出芽酵母 <i>Saccharomyces cerevisiae</i>	最小のモデル真核生物	ブドウ果皮、ビール	12,069	約 6300
シロイヌナズナ <i>Arabidopsis thaliana</i>	顕花植物のモデル生物	土壌と大気	約 142,000	約 26,000
線虫 <i>Caenorhabditis elegans</i>	発生を完全に記載できる単純な動物	土 壤	約 97,000	約 20,000
キイロショウジョウバエ <i>Drosophila melanogaster</i>	動物発生の遺伝学に貢献	腐りかけの果物	約 137,000	約 14,000
ヒト <i>Homo sapiens</i>	最も精力的に研究されている哺乳類	家	約 3,200,000	約 24,000

ゲノムサイズや遺伝子数は、特に細菌と古細菌の場合、同じ種でも系統によって異なる。表のデータは配列決定された特定の系統のもの。遺伝子には何通りものタンパク質を生じるものが多いので、ゲノムによって規定されるタンパク質の総数は遺伝子数よりかなり多い。

## ショットガン シークエンス法





ボックス 1.2 DNA分子の長さの表し方

5

DNAは二本鎖なので、塩基対(base pair; bp)の数で分子の長さを表す。キロ塩基対(kilobase pair; kb)は $10^3$  bp、メガ塩基対(megabase pair; Mb)は $10^6$  bp、ギガ塩基対(gigabase pair; Gb)は $10^9$  bp。まとめると、  
 $1 \text{ kb} = 1000 \text{ bp}$   
 $1 \text{ Mb} = 1000 \text{ kb} = 1,000,000 \text{ bp}$   
 $1 \text{ Gb} = 1000 \text{ Mb} = 1,000,000 \text{ kb} = 1,000,000,000 \text{ bp}$   
 RNA分子はたいてい一本鎖なので長さの単位にbpは使えず、ヌクレオチドの数で表す。



- 1995年、生物として初めて*Haemophilus influenzae*の全ゲノムが解読された
- その後、多くの生物の全ゲノムが解読され、現在では3000種以上の生物のゲノム情報がデータベースに登録されている

ヒトゲノムマップを開く

6

<http://www.lif.kyoto-u.ac.jp/genomemap/>

ゲノムブラウザを使ってヒトゲノムを見てみよう

7

NCBIトップページ右のリンクから「Genome」→「Genome Data Viewer」

ゲノム解読された真核生物の系統図が表示される  
 「human」  
 ↓  
 「Browse genome」をクリック

クリックした染色体のゲノムマップが表示される

青い領域にコードされる遺伝子が下に表示される

黒い領域は塩基配列が決まっていない

遺伝子マップ  
 太線: エキソン  
 細線: インtron  
 遺伝子発現量

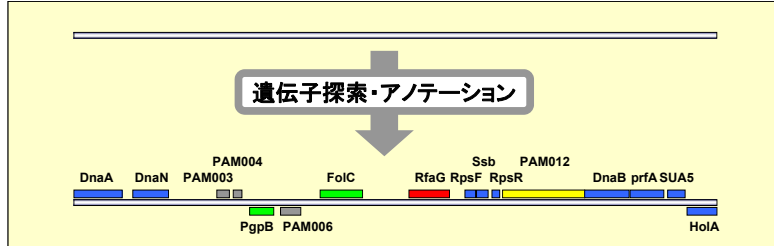
「ABO」と入力してみる

セントロメアには遺伝子がコードされていない

8

# 遺伝子探索

agttatTTTTTTTcattttataaattaaagccaatttaaaaaaggagatttaattatgccat  
 atattgaaagtatttttagcgcgcgaagtgcagattccagaggaaatcctacagtagaagtaga  
 agttatacagaatcaggagcgtttggaagagctattgttccttcaggagccttaccggacaa  
 tacgaagcagttgaattaagggatgggatgcccaaagatttttaggtaaaggcgttttgcaag  
 ctgttaaaaatgttattgaagtattcaaccagaattagaaggttattctgtcttagaacaaa  
 ttaattgataaattatataaacttgacggaactcctaacaaaatctaatttaggagctaac  
 gctatttttagtggtttctttggcttggctaaagctgcagctaacacttaaatcttgagtttt  
 atcaatatgtaggaggcgttttacctaacaacaaatgccagttcctatgatgaatttatcaacgg  
 tggagc.....

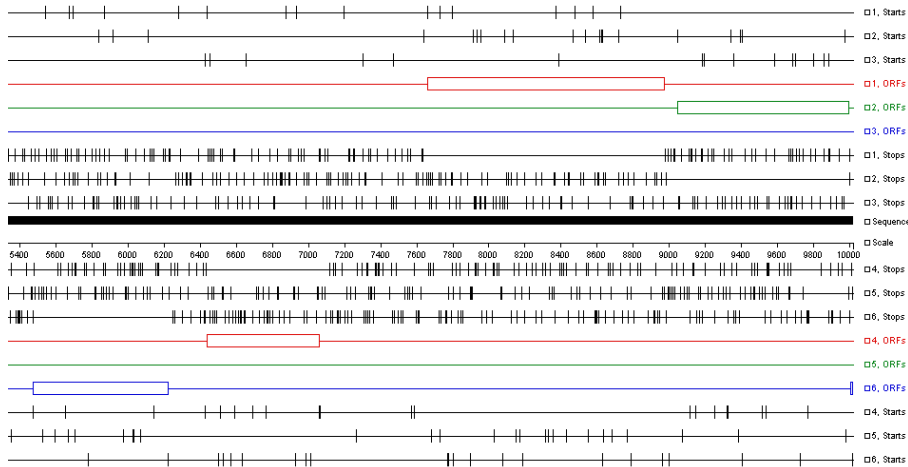
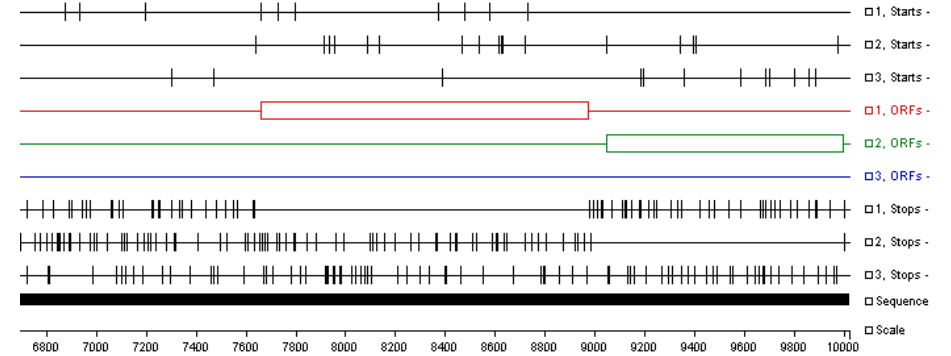


ゲノム配列から生命活動に関わる機能や分子進化に関する考察などを行うためには、タンパク質をコードしている遺伝子領域を同定することが重要となる。



図 7.1 二本鎖 DNA 分子には 6 種類の読み枠がある

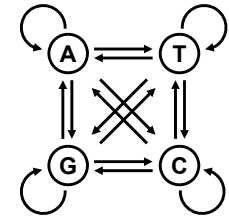
両鎖とともに、5'→3'方向に読まれる。どのヌクレオチドを開始位置として選ぶかによって、どちらの鎖にも 3 つの読み枠がありうる。



# GeneMark.hmm

<http://opal.biology.gatech.edu/GeneMark/>

隠れマルコフモデルを用いた遺伝子検出プログラム



DNA配列の1次マルコフモデル

■ マルコフモデルとは、ある記号の出現確率が、直前のm個の記号によって決定されるような確率モデル

■ タンパク質コード領域では、コドン3文字のそれぞれの位置での塩基の出現頻度が異なるため、この特徴を利用して遺伝子を検出する

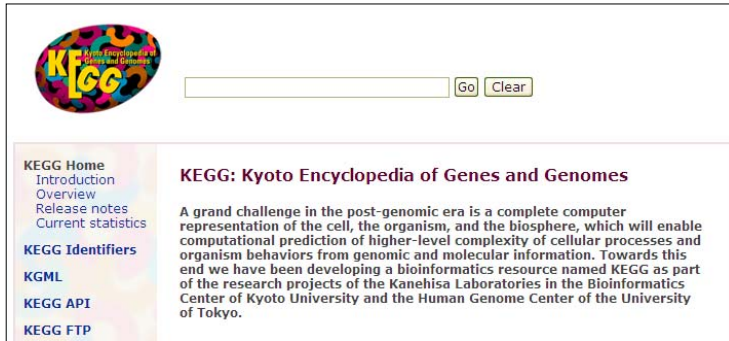
CTTATAGGAGAATAAAAAGATGGGTCGTCAGCCTTCAATGGAA  
 M G R Q P S M E

■ 例えばGeneMarkでは、コード領域をモデル化するために3個の異なる5次マルコフモデルを用いている。(ある文字の出現確率は前の5個の文字に依存するというモデル)

# 代謝パスウェイデータベース

KEGG

http://www.genome.jp/kegg/



生命システム情報統合データベース。完全にゲノムが決まった生物種(一部、ドラフト配列も含む)の代謝系や一部の制御系(シグナル伝達や細胞周期など)をまとめ、そこから様々な物質データベースや酵素データベースを参照することができる。

# 代謝系データベースの参照

生物種を選んで「Go」を押す

選んだ生物のゲノムにコードされている酵素にだけ色がつく。例えば「Rickettsia prowazekii」を選ぶと、ほとんど色がつかず、解糖系を持っていないことがわかる。

# 次世代シーケンサー

Roche Diagnostics社  
**Genome Sequencer FLX System** (454)  
 2005年発売



ライフテクノロジー社  
**Ion PGM**



Applied Biosystems社  
**SOLID 3**  
 2007年発売



Solexa / illumina社  
**Genome Analyzer I/x**  
 2005発売



イルミナ株式会社  
**MiSeq**



PacBio RS II

# 次世代シーケンサーの比較

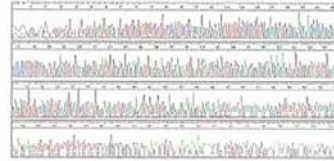
	Ion Protonシステム	Ion PGMシステム Ion 318 chip	MiSeq	HiSeq 2000/2500 (SBS v3試薬使用)	PacBio RS II
1リード長	~200 base		150/250/300 base	100 base	約10,000 base
リード数	約5,000万リード (1ランあたり)	約400万リード (1ランあたり)	約3,000万リード (1ランあたり) ※ペアエンド解析	約3億リード (1レーンあたり) ※ペアエンド解析	約5万リード (1セルあたり)
データ量 (リード長 200 base の場合)	約7.5 Gb (平均150 bpの amplicon、 1チップあたり)	約800 Mb (1チップあたり)	約3~9 Gb (1ランあたり) ※ペアエンド解析	約30 Gb (1レーンあたり) ※ペアエンド解析	約500 Mb (1セルあたり)
解析手法	Ion semiconductor sequencing法		Sequencing by Synthesis法	Sequencing by Synthesis法	SMRT(Silgle Molecule Real-Time) sequencing法
アプリケーション例	・癌遺伝子などの変異解析 (409遺伝子をターゲットとしたCancer Panelなど)	・癌遺伝子などの変異解析 (50遺伝子をターゲットとしたCancer Panelなど)	・微生物の新規ドラフト配列決定 ・癌遺伝子などの変異解析 ・PCR産物のディープシーケンス	・ゲノム変異解析 ・ChIP解析 ・small RNA解析 ・mRNA解析 ・cDNA配列解析	・ゲノムドラフト解析 ・cDNA配列解析

次世代シーケンサー: Genome Analyzer



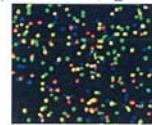
● 従来型キャピラリーシーケンサー

- 酵素反応+電気泳導+塩基読取 (384x600塩基)
- コスト、時間がかかる
- 例)「ヒトゲノムプロジェクト」  
約13年、3000億円かかった

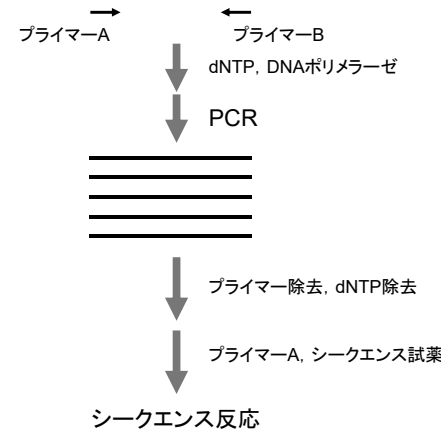
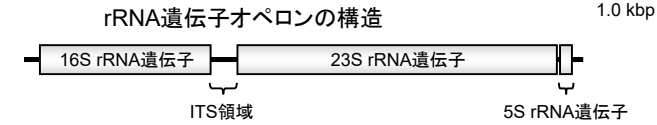


● 次世代シーケンサー

- 酵素反応+電気泳導+塩基読取 (100,000,000x50塩基)
- これまでの技術と比べて、「100分の1のコストで100倍のデータ」
- 例) 現在ヒトゲノム1人読むのに 数週間、数千万円  
→ 1週間 数百万円 ...

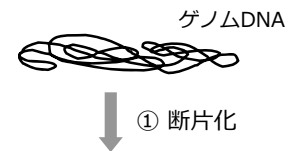


illumina

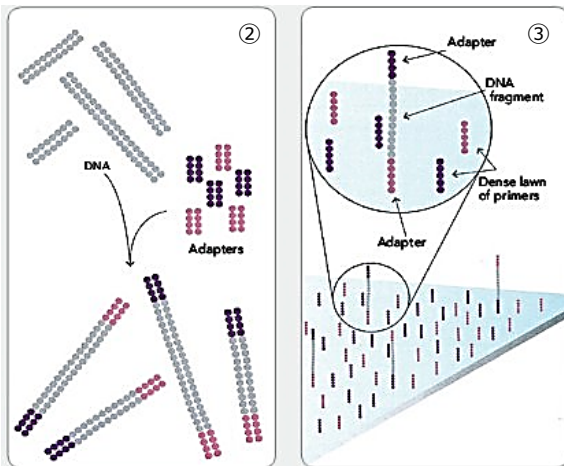


- ◆ PCR産物をシーケンスする方法も良く使われる
- ◆ PCRでDNAを増幅するので、大腸菌を使わなくても済む
- ◆ しかし、この方法だと多サンプルをシーケンスするのが困難
- ◆ また、塩基配列が一部わかっている場合しかこの方法は使えない

次世代シーケンサーの原理 1 サンプル調製 ~ フローセルへの固定 19



- ゲノムDNAを抽出し、断片化する
- DNA断片の両端に、2種類の **アダプター** (アダプター-1、2)を連結させる

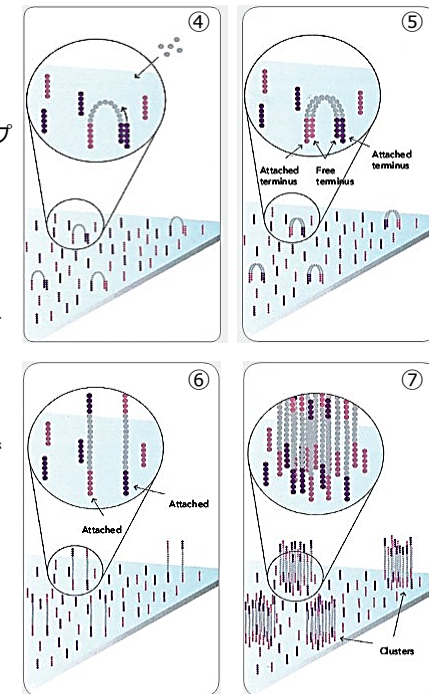


- 1本鎖にして、5'末端(アダプター-1の側)を **フローセル** 上に固定する
- フローセル上には、あらかじめアダプター-1、2と相補的に結合するプライマーが高密度に配置されている

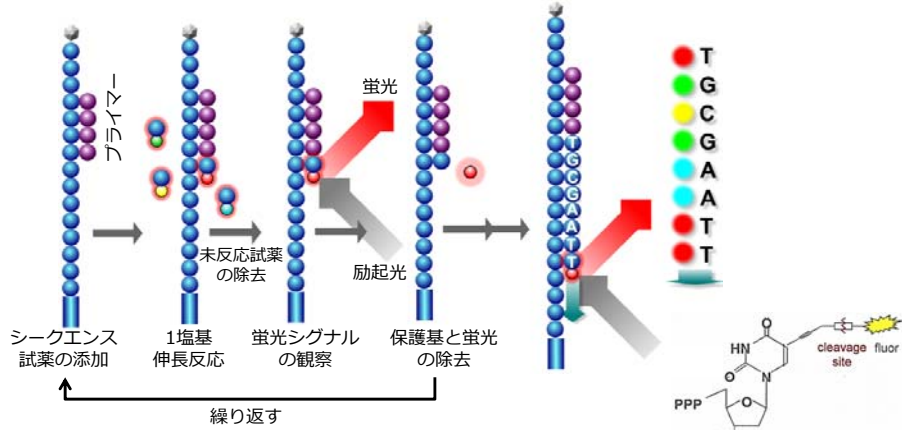
次世代シーケンサーの原理 2

ブリッジPCR

- 固定された1本鎖DNAはアダプター-2の側でプライマーと結合する(橋がかかったような構造になる ④)
- DNAポリメラーゼによる伸長反応を行う ⑤
- 変性させると、フローセル上にはアダプター-1側で結合した1本鎖と、アダプター-2側で結合した1本鎖ができあがる ⑥
- この反応を繰り返すことで、狭い面積の中でDNAを増幅することができる  
→ フローセル上に多数のDNAの「束」ができる ⑦
- これらを鋳型として、配列解析を行う



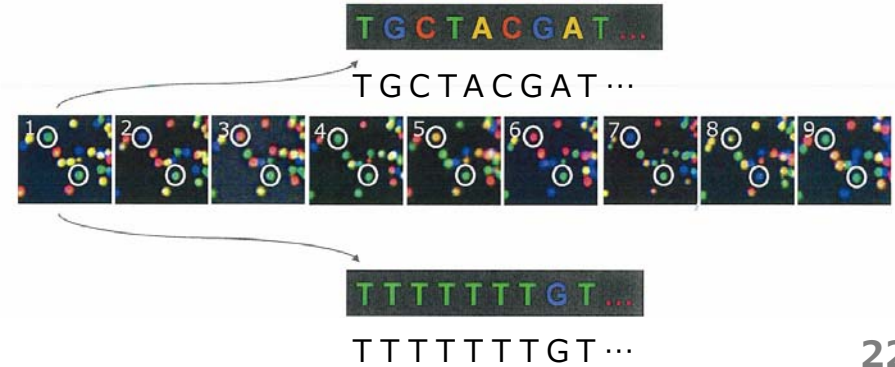
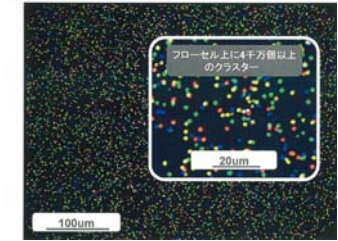
Sequencing-by-synthesis による塩基配列決定



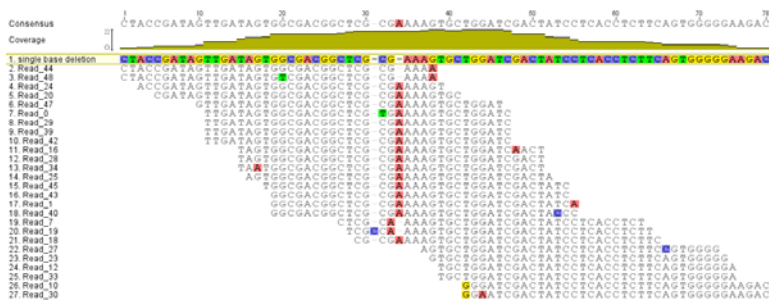
- 蛍光標識したdNTPの取り込みを蛍光顕微鏡によって解析する
- このdNTPは3'末端がブロックされており、1回の伸長反応で1塩基しか伸ばせない
- そのため、1塩基ごとにどのdNTPが取り込まれたかを観察し、蛍光物質とブロックを外して次の伸長反応を行うというステップで、解析を進めていく

画像蛍光シグナルから塩基への返還

- 1塩基 伸長するごとに蛍光イメージを取得する
- それぞれのDNAの「束」の蛍光色の変化を調べることで、塩基配列を決定する
- 数千万～数億本の塩基配列が得られる



- ◆ 一つ一つの断片の塩基配列が短いと、アセンブルするのが困難
- ◆ 次世代シーケンサーで読み取ることができる塩基配列長は、未だ短いので、既に全塩基が解読されているゲノム配列 (リファレンス配列) を利用したリシーケンスや、リファレンス配列へのマッピングなどに用いられることが多い



マッピング : Bowtie, Bowtie2, BWA など  
 アセンブル : Velvet, EDENA, Phrap など  
 ビューア : Tablet, IGV など

有償ではあるが、CLC Genomics Workbenchなどの解析ソフトも良く使われる

*Pectobacterium carotovorum* sp. *carotovorum*



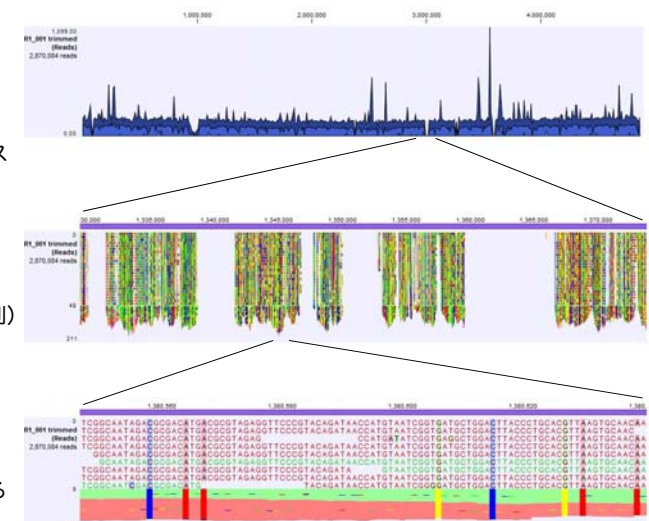
*Pectobacterium carotovorum* PR1 strain



*P. carotovorum* PR1株のゲノムを抽出

MiSeqを用いてシーケンス (約300万リード)

*P. carotovorum* sp. *carotovorum* のゲノム (リファレンス配列) に対してマッピング



- 遺伝子の有無
  - ゲノム構造の比較
  - SNPの検出
- 等の比較ゲノム解析ができる

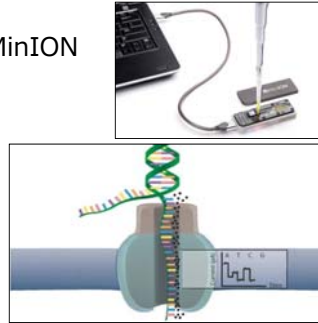
### 第3世代シーケンサー

#### Pacbio RS II DNA Sequencing System



- DNA 1分子 を鋳型としてDNAポリメラーゼによるDNA合成を行う
- 1分子レベルでリアルタイムに塩基を読み取る
- 長いリード(平均10,000bp)が出力される

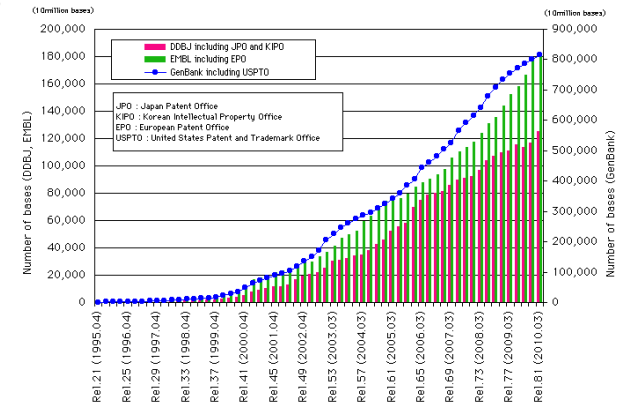
#### MinION



- USBメモリー用のシーケンサー
- DNAポリメラーゼを用いて1本鎖DNAに解きほぐす
- ナノポア を通過させる → 電流の変化を検知して配列を決定する

### 塩基データ登録数の推移

- シークエンス技術の進歩によって、塩基配列決定の速度はますます加速している



- 遺伝子の検出, アノテーション, 機能予測, 進化系統解析, 比較解析などを効率よく行い, 大量のシーケンスデータを有効に活用することが重要

ゲノムにコードされる遺伝子を網羅的に使用してホモロジー検索を行ったり, 比較ゲノム解析を行いたい



大量のデータを処理するためのプログラミング技術が必要

バイオインフォマティクス分野では, Perl, C++, Java, Pythonなどが良く使われていますが, 本日はPerlを用いて実習を行います



#### Perlの特徴

- テキスト処理が得意
- 歴史が長いのでライブラリーが豊富
- 掲示板やショッピングカートなど, CGIという仕組みの多くがPerlで書かれている
- LinuxやMacOSに標準でインストールされている (ほか, Windowsにもインストール可能)

#### プログラミング言語の人気ランキング (2016)

- 【第1位】 Java
- 【第2位】 C言語
- 【第3位】 C++
- 【第4位】 Python
- 【第5位】 C#
- 【第6位】 Visual Basic
- 【第7位】 JavaScript
- 【第8位】 PHP
- 【第9位】 Perl
- 【第10位】 アセンブリ言語

### perlを用いたデータ処理

- 大量のQueryに対してBLAST検索を行うと, 結果が羅列した形で出力されます
- この中から, 必要な情報だけを取りだしてくるためのプログラムをperlで組んでみましょう
- Queryのアクセス番号と, 検索の結果ヒットしたタンパク質のアクセス番号とのリストを作成し, 下に示したように検索結果を整理したいと思います

Query	1	2	3	4	5
gi 49176138 ref NP_416237.3	ref NP_009965.1	ref NP_010402.1	ref NP_012405.1	ref NP_012934.1	ref NP_015093.1
gi 16132212 ref NP_418812.1	ref NP_014926.1	ref NP_012380.1	ref NP_012969.1	ref NP_012770.1	ref NP_014585.1
gi 16131951 ref NP_418449.1	ref NP_009755.1	ref NP_011646.1	ref NP_013146.1	ref NP_013847.1	ref NP_013523.1
gi 16131757 ref NP_418354.1	ref NP_010335.1	ref NP_012586.1			
gi 16131754 ref NP_418351.1	ref NP_011756.1	ref NP_013932.1	ref NP_010104.1	ref NP_011671.1	
gi 16131018 ref NP_417595.1	ref NP_009362.1	ref NP_014892.1	ref NP_014792.1	ref NP_014227.1	ref NP_011015.1
gi 16130827 ref NP_417401.1	ref NP_009938.1	ref NP_011705.1	ref NP_011575.1	ref NP_011569.1	ref NP_012819.1
gi 16130826 ref NP_417400.1	ref NP_012863.1	ref NP_013282.1	ref NP_015308.1	ref NP_012835.1	ref NP_010022.1
gi 16130686 ref NP_417259.1	ref NP_011770.1	ref NP_012044.1	ref NP_014056.1	ref NP_015042.1	ref NP_015038.1
gi 16130106 ref NP_416673.1	ref NP_009965.1	ref NP_014276.1	ref NP_012639.1	ref NP_013060.1	ref NP_013066.1

BLASTP 2.2.5 [Nov-16-2002]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

Query= gi|16131851|ref|NP\_418449.1| glucosephosphate isomerase  
[Escherichia coli K12]  
(549 letters)

Database: yeast.aa  
6298 sequences; 2,974,038 total letters

Sequences producing significant alignments:	Score (bits)	E Value
ref NP_009755.1  Glucose-6-phosphate isomerase; Pgilp	641	0.0
ref NP_011646.1  Ygr130cp	30	0.98
ref NP_013146.1  spindle pole body component; Stu2p	29	1.7
ref NP_013847.1  (putative) involved in cell wall biogenesis; Ec...	28	3.7
ref NP_013523.1  Ylr419wp	28	3.7

>ref|NP\_009755.1| Glucose-6-phosphate isomerase; Pgilp  
Length = 554

Score = 641 bits (1654), Expect = 0.0  
Identities = 326/549 (59%), Positives = 401/549 (73%), Gaps = 16/549 (2%)

Query: 7 TQTAAWQALQKHFDDEM-KDVTIADLFAKDGDRFSKFSATFDD---QMLVDYSKNRITEE 61  
T+ AW LQK ++ K +++ F KD RF K + TF + ++L DYSKN + +E  
Sbjct: 13 TELPAWSKQLKIYESQKTLVSKQEFQKDAKRFKLNKFTNYDGSKILFDYSKNLVNDE 72

Query: 62 TLAKLQDLAKECDLAGAIKSMFSGEKINRTENRAVLHVALRNRNTPILVDGKDVMPVEVN 121  
+A L +LAKE ++ G +MF GE IN TE+RAV HVALRNR+N P+ VDG +V PEV+  
Sbjct: 73 IIAALIELAKEANVTGLRDMFQGEHINSTEDRAVYHVALRNRANKPMYVDGNNVAPEVD 132

- ◆ デスクトップ上に、「kiso」フォルダを作成してください  
(既に「kiso」フォルダがある場合は削除して、新たに作成してください)

## 1. 生物配列解析基礎

### 授業の目標・概要

生命科学のためのデータベースの利用と基本的な解析手法について講義します。データベースの基礎、配列データベース、機能データベース、ホモロジー検索、モチーフ解析などの基本的な手法について解説します。

kiso2

blast.pl


result.txt

result.txt

blast.pl

の2つのファイルをダウンロードして、kisoフォルダに入れてください

- ◆ コマンドプロンプトを立ち上げてください

 スタート → すべてのプログラム → アクセサリ → コマンドプロンプト

まず、kisoフォルダに移動します

```
> cd \
```

「cd(スペース)」と入力した後(まだEnterキーは押さない)、kisoフォルダをコマンドプロンプト上にドラッグ&ドロップしてください

下記のように表示されますので、Enterキーを押してください

```
> cd C:\Users\%iu\Desktop\kiso
```

- ◆ blast.plをメモ帳を使って開いてください

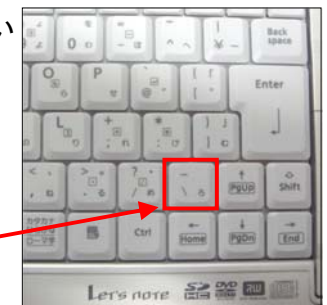
```
#! /usr/local/bin/perl
```

- ◆ 以下のように編集して、上書き保存してください

```
#! /usr/local/bin/perl
print "Hello!\n";
```

¥nは改行を表します

「¥」は、バックslash「\」を押してください  
Windows上だと「¥」と表示されます



以下のコマンドを入力して、プログラムを実行してください

```
> perl blast.pl
```



## 変数

- ◆ 変数は、「\$文字列」で表します
- ◆ 以下のように編集してください

```
#!/usr/local/bin/perl
$a = "Hello!¥n";
print $a;
```

「;」を入力し忘れないように注意してください。Perlでは行の終わりに「;」をつける決まりになっています

以下のコマンドを入力して、プログラムを実行してください

```
> perl blast.pl
```

## <STDIN>

- <STDIN>は、1回呼び出すごとに標準入力から1行のデータを読み出す命令です。STDINとは、standard inputつまり標準入力の略です。
- 以下のプログラムを作成してください。

```
blast.pl
```

```
#!/usr/local/bin/perl
$a = <STDIN>;      # 標準入力から1行のデータを読み出し、$aに代入する
print $a;         # $aを出力する
```

- 以下のコマンドを打ち込みblast.plを実行すると、入力待ちになります。何か文字を入力すると、入力した文字がそのまま出力されます。

```
> perl blast1.pl
```

## リダイレクト

- 標準入力を用いる際に、キーボードから文字列を打ち込む代わりに、既存のテキストファイルからデータを読み込ませることもできます。
- 「result.txt」というBLAST検索結果のファイルを用意しておきました。中身を見てください。

```
BLASTP 2.2.19 [Nov-02-2008]
```

```
Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.
.
```

## リダイレクト

- 「result.txt」を読みこませるために、リダイレクトという機能を使います。

```
> perl blast.pl < result.txt
```

「result.txt」の一番最初の行だけが表示されます。

- 結果を、画面でなく、ファイルに出力することもできます。

```
> perl blast.pl < result.txt > output.txt
```

## whileループ

- 読み込むデータが何行であっても良いように、whileループを利用してみましょう。

```
#!/usr/local/bin/perl
while ($a = <STDIN>) { # データを1行ずつ$aに
    print $a;          # 代入して、出力する
}
```

読み込むデータ(行)がある限り「真」となり、繰り返されます。

- 実行してみましょう。全てのデータが表示されるはずです。

```
> perl blast.pl < result.txt > output.txt
```

## 演算子「=~」

- 文字列「DNA」を含む行を表示してみましょう。

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
    if ($a =~ /DNA/) { # DNAを見つけたら
        print $a;      # 出力する
    }
}
```

「~」チルダ  
Shiftを押しながら



- 「=~」をパターン結合演算子、「/文字列/」をマッチ演算子といいます。文字列の一部に一致すれば「真」を、一致しなければ「偽」を返します
- \$a =~ /DNA/ は、\$aにDNAという文字列が含まれていれば「真」となり、if文の中身が実行されます。

## 練習

- 文字列「Query=」を含む行を表示してみましょう。

## 正規表現による検索

- /と/の間には、文字列だけでなく、パターンと呼ばれるものを入れることができます。ここでパターンとは、「Mで始まる文字列」や「3文字の文字列」など、固定の文字列の代わりに文字列の特徴を記述したものです。このパターンの記述方法を正規表現といいます。

例えば、

DNA

DNNA

DNNNNNA

DNNNNNNNNNNNNNNNNNA

これらすべてを検索するには、

/DN+A/

と記述します



## 位置指定

パターンの位置を指定します。

<code>^</code>	先頭
<code>\$</code>	末尾

## エスケープ

/、^、\$などの、正規表現的に意味のある特殊記号自体を検索したい局面では、`\`でエスケープします。

<code>^\<code>^</code></code>	<code>^</code> という字で始まる行にマッチ
<code>\\</code>	<code>\</code> 自体にマッチ

## 文字クラス

以下のものは、次のような1文字にマッチします。

<code>[abc]</code>	aかbかcのどれか
<code>[a-z]</code>	任意の小文字
<code>[^abc]</code>	aでもbでもcでもない文字
<code>\d</code>	数字 (digit)
<code>\D</code>	数字以外
<code>\w</code>	英数字 (word)
<code>\W</code>	英数字以外
<code>\s</code>	空白文字 (space)
<code>\S</code>	空白文字以外
<code>\b</code>	単語境界 (word boundary)
<code>.</code>	任意の1文字

41

## 繰り返し

以下の記号を使って、文字または文字クラスの繰り返しとマッチします。ここでは文字または文字クラスをxと書きます。

<code>x*</code>	0回以上の繰り返し
<code>x+</code>	1回以上の繰り返し。xx*と同じ
<code>x?</code>	0回か1回
<code>x{5}</code>	5回繰り返し。xxxxxと同じ
<code>x{3,}</code>	3回以上繰り返し。xxx+と同じ
<code>x{3,5}</code>	3回以上5回以下繰り返し。xxx?x?と同じ

## グループと選択

文字列を繰り返すときは()を使ってグループ化します。

`su(mo)+` sumo, sumomo, sumomomoなどにマッチする

いくつかのパターンのどれかにマッチさせるときは|を使います。

<code>love!kiss</code>	loveかkissにマッチする
<code>stud(y!ies)</code>	studyかstudiesにマッチする
<code>su(mi!mo){2,3}</code>	sumimi, sumimo, sumomi, sumomo, sumimimi, sumimimo, sumimomi, sumomimi, sumomomi, sumomimo, sumimomo, sumomomoのいずれかにマッチする

42

43

## 正規表現による検索

- Gene indexを含む文字列を抽出してみましょう。

```
Query= gi|13507742|ref|NP_109691.1| DNA gyrase
```

「Query=」と「ref」ではさまれた連続した文字列を含む行を抽出するには・・・

「.」(任意の文字) と「+」(1文字以上の連続文字) を使って以下のようにします

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
    if ($a =~ /Query= .+ref/) {
        print $a;
    }
}
```

44

## カッコを使った記憶

- マッチ演算子のパターンの中で括弧()を使うと、その括弧で囲まれた部分が、\$1, \$2, ...という特殊変数に格納されます。

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
    if ($a =~ /Query= (.+)ref/) {
        print $1;
    }
}
```

## 改行

- 改行されるように, “`\n`”を入れます

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "\n",$1;
  }
}
```

- BLAST検索の結果, ヒットしたタンパク質の情報 (例えば  
`>ref|NP_072866.1| topoisomerase IV, subunit A)`  
 を含む行を抽出し, タブ区切りで表示してみましょう

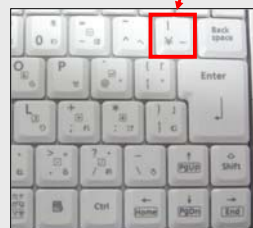
```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "\n",$1;
  }
  if ($a =~ />ref/) {
    print "\t",$a;
  }
}
```

- ヒットしたタンパク質情報のref番号だけを抽出してみましょう  
`>ref|NP_072866.1| topoisomerase IV, subunit A)`

“|”ではさまれた文字列を取り出したいのですが, 以下の表現ではうまくいきません

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "\n",$1;
  }
  if ($a =~ />ref|.+/) {
    print "\t",$a;
  }
}
```

「|」  
Shiftを押しながら



“|”は正規表現で使用する特殊な文字であるため, 別の意味になってしまうからです

ここで使う “|” が正規表現でないことを示すために, `\|`を頭につけます

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "\n",$1;
  }
  if ($a =~ />ref\|.+\/) {
    print "\t",$a;
  }
}
```

- 括弧を使って、ref番号だけを抽出してみましょう

```
>ref|NP_072866.1| topoisomerase IV, subunit A)
```

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "%n",$1;
  }
  if ($a =~ />ref¥!(.+)¥!/) {
    print "%t",$1;
  }
}
```

QueryのGene Index

ヒットしたタンパク質のref番号

```
gi|13507740| NP_072661.1
gi|13507741| NP_072662.1
gi|13507742| NP_072663.1 NP_072865.1
gi|13507743| NP_072664.1 NP_072866.1
gi|13507744| NP_072665.1
gi|13507745| NP_072666.1
gi|13507746| NP_072667.1
gi|13507747| NP_072668.1 NP_072998.1
gi|13507748| NP_072669.1
.
.
.
```

- 最も似ている配列の情報だけを表示するようにしてみましょう
- \$b という変数が0か1かを指標として、Queryを見つけた直後のヒット情報だけを取り出します

```
#!/usr/local/bin/perl
while ($a = <STDIN>) {
  if ($a =~ /Query= (.+)ref/) {
    print "%n",$1;
    $b = 1;
  }
  if ($a =~ />ref¥!(.+)¥!/ && $b == 1) {
    print "%t",$1;
    $b = 0;
  }
}
```

QueryのGene Index

ヒットしたタンパク質のref番号

```
gi|13507740| NP_072661.1
gi|13507741| NP_072662.1
gi|13507742| NP_072663.1
gi|13507743| NP_072664.1
gi|13507744| NP_072665.1
gi|13507745| NP_072666.1
gi|13507746| NP_072667.1
gi|13507747| NP_072668.1
gi|13507748| NP_072669.1
gi|13507749|
gi|13507750|
gi|13507751|
gi|13507752|
gi|13507753| NP_072670.1
gi|13507754| NP_072671.1
.
.
.
```

## <課題1>

- QueryのGene Indexのうち、数字の部分だけを取り出し、以下のような出力結果になるようなプログラムを作成してください

```

13507740 NP_072661.1
13507741 NP_072662.1
13507742 NP_072663.1
13507743 NP_072664.1
13507744 NP_072665.1
13507745 NP_072666.1
13507746 NP_072667.1
13507747 NP_072668.1
13507748 NP_072669.1
.
.

```

## <課題2> 余裕のある方は…

- E-valueの値を取り出し、以下のような出力結果になるようなプログラムを作成してください

Queryの Gene Index	ヒットしたタンパク質 のref番号	E-value
13507740	NP_072661.1	e-148
13507741	NP_072662.1	e-125
13507742	NP_072663.1	0.0
13507743	NP_072664.1	0.0
13507744	NP_072665.1	0.0
13507745	NP_072666.1	1e-077
.	.	.

課題1と同じファイル名にならないように、blast2.plやoutput2.txtなどにファイル名を変えてください

## <課題の提出方法>

- 出力したテキストファイル(output)を提出してください

「受講生の方へ」のページ



「課題提出用Web mailページへ(講義室のみからアクセス可)」

送信先:	kenro@ims.u-tokyo.ac.jp	← kenro@hosei.ac.jpを選ぶ
件名:	Perl課題	← 「Perl課題」と入力
氏名:		「氏名」「所属」「学生証番号」「メールアドレス」を入力
所属:		
学生証番号:		
E-mail:	<input type="checkbox"/> Ccを送る	
本文:	<div style="border: 1px solid gray; height: 150px; width: 100%;"></div> <p>← 本日の講義の感想などを、ご記入ください</p>	
添付ファイル: 次の確認画面で指定して下さい		
<input type="button" value="確認"/> <input type="button" value="リセット"/>		