Bio::Tools::Run::RemoteBlast

- Klasse, um bei NCBI BLAST-Anfragen zu starten und die Ergebnisse zu holen
- Version von BioPerl 1.4 nicht mehr (voll) funktionstüchtig (aber in BioPerl 1.5)
- Für die Übung gepatchte Version aus BioPerl 1.4

Bio::Tools::Run::RemoteBlast (2)

Konstruktor:

$blast = Bio::Tools::Run::RemoteBlast->new();

Argumente:

- -prog: BLAST-Version ('blastp', 'blastn', ...)
- -data: Datenbank ('swissprot', ...)
- -expect: E-Wert (Parameter von BLAST), je höher, desto wahrscheinlicher eine entsprechende Sequenzähnlichkeit auch zufällig zu finden
- -readmethod: 'SearchIO' oder 'BPlite'

-readmethod eigentlich veraltet, aber in gepatchter Version funktioniert nur BPlite ...

Bio::Tools::Run::RemoteBlast (3)

Methoden:

```
$blast->submit_blast($seq);
$blast->submit_blast(@seqs);
$blast->submit_blast( $file );

$blast->retrieve_blast ();
```

retrieve_blast () gibt entweder ein Bio::SearchIO- oder ein Bio::Tools::BPlite-Objekt zurück, wenn ein Ergebnis vorliegt, vorher -1 für 'Fehler' und 0 für 'noch nicht fertig'

Bio::SearchIO

Moderne Klasse für das Parsen von BLAST-Ergebnissen (nur leider für uns nicht benutzbar)
Konstruktor (nur für lokale Files):

```
Bio::SearchIO->new();
```

Argumente:

- -file: Ergebnisfile von BLAST
- -format: 'blast', 'blastxml', 'fasta', ...

Zwei wichtige Methoden:

```
$searchio->result_count() # Anzahl der Ergebnisse
$searchio->next_result() # naechstes Ergebnis
```

Bio::Search::Result::ResultI

Interface für einzelnes BLAST-Ergebnis
Methoden:

$result->algorithm() *#verwendeter Algorithmus (blastp ,...)*
$result->num_hits() *#Anzahl Treffer*
$result->next_hit() *#naechster Treffer*

next_hit () gibt ein Objekt des Interfaces Bio :: Search :: Hit :: HitI zurück

Bio::Search::Hit::HitI

Interface für einzelne Hits
Methoden:

$hit->name(); *#Name als Skalar*
$hit->description(); *#Beschreibung (Skalar)*
$hit->**length**(); *#Laenge (Skalar)*
$hit->significance(); *#Signifikanz als E- oder p-Wert*
$hit->next_hsp(); *#Naechstes High Scoring Pair*

Bio::Tools::BPlite

Ältere und einfachere Alternative zu Bio::SearchIO
Methoden:

$lite->database(); *#Benutzte Datenbank*
$lite->nextSbjct(); *#Naechstes 'Subject'*

nextSbjct() gibt Objekt der Klasse Bio::Tools::BPlite::Sbjct zurück

Bio::Tools::BPlite::Sbjct

Simple und veraltete Alternative zu Bio::Search::Hit::HitI mit deutlich weniger Methoden:

```
$sbjct->name(); #Name des Treffers als Skalar
$sbjct->nextHSP(); #Nachstes High Scoring Pair
```

name() liefert z.B. für BLASTP den Namen des gefundenen Proteins

Bio::Restriction::EnzymeCollection

Klasse für eine Menge von Restriktionsenzymen
Konstruktor:

$ec = Bio::Restriction::EnzymeCollection->new();

Legt neue Menge mit allen Enzymen an, die BioPerl kennt.
Methoden:

$ec->blunt_enzyme(); *#Liste der Enzyme die glatt schneiden*
$ec->each_enzyme(); *#Liste aller Enzyme*
$ec->get_enzyme($name); *#Enzym mit bestimmtem Namen*

get_enzyme() liefert ein Bio::Restriction::Enzyme-Objekt zurück

Bio::Restriction::Analysis

Klasse für die Analyse von Restriktionsschnittstellen
Konstruktor:

```
$an = Bio::Restriction::Analysis->new();
```

Argumente:

- -seq: Bio::Seqlobjekt das geschnitten werden soll
- -enzymes: (optional) Bio::Restriction::EnzymeCollection-Objekt

Methoden:

```
$an->fragments($enz) #Berechnet die Fragmente fuer ein bestimmtes Enzym,
                                    #Rueckgabe ist Liste von Skalaren (Strings)
$an->sizes($enz) #Groesse der Fragmente (funktioniert in 1.4 nicht)
...
```

The End