G.pm (Prelude) System Core

of G-language Genome Analysis Environment Version 1

1. AUTHOR

Kazuharu Arakawa, gaou@g-language.org

2. LAST UPDATE OF THIS DOCUMENT

March 21, 2002

3. INTRODUCTION

G.pm (Prelude) is the core module of G-language Genome Analysis Environment. By importing this module from a Perl program, all method included in the software package will be made available. When a genome database is loaded using this module, it will be stored in memory in a unique style specific to G-language GAE. This documentation describes the structure of the loaded database, and the basic methods provided by this module to manipulate the genome database.

4. SYNOPSIS

```
use G;                          # Imports G-language GAE module

$gb = new G("ecoli.gbk");       # Creates G's instance at $gb
                                # At the same time, read in ecoli.gbk.
                                # Read the annotation and sequence information
                                # See DESCRIPTION for details

$gb->seq_info();                # Prints the basic sequence information.

$find_ori_ter(¥$gb->{SEQ});     # Gives sequence as a reference to
                                # odyssey functions
```

5. DESCRIPTION

The Prelude Core of G-language GAE fully supports most sequence databases.

Stored annotation information:

LOCUS

$gb->{LOCUS}->{id}	-accession number
$gb->{LOCUS}->{length}	-length of sequence
$gb->{LOCUS}->{nucleotide}	-type of sequence ex. DNA, RNA
$gb->{LOCUS}->{circular}	-1 when the genome is circular. otherwise 0
$gb->{LOCUS}->{type}	-type of species ex. BCT, CON
$gb->{LOCUS}->{date}	-date of accession

HEADER

$gb->{HEADER}

COMMENT

$gb->{COMMENT}

FEATURE

Each FEATURE is numbered(FEATURE1 .. FEATURE1172), and is a hash structure that contains all the keys of Genbank.
In other words, in most cases, FEATURE$i's hash at least contains informations listed below:

$gb->{FEATURE$i}->{start}
$gb->{FEATURE$i}->{end}
$gb->{FEATURE$i}->{direction}
$gb->{FEATURE$i}->{join}
$gb->{FEATURE$i}->{note}
$gb->{FEATURE$i}->{type} -CDS,gene,RNA,etc.

To analyze each FEATURE, write:

```
$i = 1;
while(defined(%{$gb->{FEATURE$i}})){
        $i ++;
}
```

Each CDS is stored in a similar manner.
There are
$gb->{CDS$i}->{start}
$gb->{CDS$i}->{end}
$gb->{CDS$i}->{direction}
$gb->{CDS$i}->{join}
$gb->{CDS$i}->{feature} -number $n for $gb->{FEATURE$n}
where "CDS$i" = "FEATURE$n"

In the same manner, to analyze all CDS, write:

```
$i = 1;
while(defined(%{$gb->{CDS$i}})){
        $i ++;
}
```

BASE COUNT

$gb->{BASE_COUNT}

SEQ

$gb->{SEQ} -sequence data following "ORIGIN"

Supported methods of Prelude

new()

Creates a G instance.
First option is the filename of the database. Default format is
the GenBank database.
Second option specifies detailed actions.

'without annotation' this option skips the annotation.

'multiple locus' this option merges multiple loci in the
database and load the information
as G-language instance.

'long sequence' this option uses a pointer of the filehandle to read the genome sequence. See next_seq() method below for details.

'bioperl' this option creates a G instance from a bioperl object.
eg. $bp = $bp->next_seq(); # bioperl
$gb = new G($bp, "bioperl"); # G

'longest ORF annotation' this option predicts genes with longest ORF algorithm (longest frame from start codon to stop codon, with more than 17 amino acids) and annotates the sequence.

'glimmer annotation' this option predicts genes using glimmer2, a gene prediction software for microbial genomes available from TIGR.
http://www.tigr.org/softlab/
Local installation of glimmer2 and setting of PATH environment value is required.

- following options require bioperl installation -

'Fasta' this option loads a Fasta format database.
'EMBL' this option loads a EMBL format database.
'swiss' this option loads a swiss format database.
'SCF' this option loads a SCF format database.
'PIR' this option loads a PIR format database.
'GCG' this option loads a GCG format database.
'raw' this option loads a raw format database.
'ace' this option loads a ace format database.
'net GenBank' this option loads a GenBank format database from NCBI database. With this option, the first value to pass to new() function will be the accession number of the database.

output()

Given a filename and an option, outputs the G-language data object
to the specified file in a flat-file database of a given format.
The options are the same as those of new(). Default format is 'GenBank'.
eg. $gb->output("my_genome.embl", "EMBL");
$gb->output("my_genome.gbk");
#with GenBank you can ommit the option.

complement()

Given a sequence, returns its complement.
eg. complement('atgc'); returns 'gcat'

translate()

Given a sequence, returns its translated sequence.
Regular codon table is used.
eg. translate('ctggtg'); returns 'LV'

$gb->seq_info()

Prints the basic information of the genome to STDOUT.

$gb->DESTROY()

Destroys the G instance

$gb->del_key()

Given a object, deletes it from the G instance structure
eg. $gb->del_key('FEATURE1'); deletes 'FEATURE1' hash

$gb->getseq()

Given the start and end positions (starting from 0 as in Perl),
returns the sequence specified.
eg. $gb->getseq(1,3); returns the 2nd, 3rd, and 4th nucleotides.

$gb->get_gbkseq()

Given the start and end positions (starting from 1 as in Genbank), returns the sequence specified.

eg. $gb->get_gbkseq(1,3); returns the 1st, 2nd, and 3rd nucleotides.

$gb->get_cdsseq()

Given a CDS ID, returns the CDS sequence.

'complement' is properly parsed.

eg. $gb->get_cdsseq('CDS1'); returns the 'CDS1' sequence.

$gb->get_geneseq()

Given a CDS ID, returns the CDS sequence, or the exon sequence If introns are present.

'complement' is properly parsed, and introns are spliced out.

eg. $gb->get_geneseq('CDS1'); returns the 'CDS1' sequence or exon.

$gb->feature()

Returns the array of all feature object name.

```
foreach ($gb->feature()){
    $gb->get_cdsseq($_);
}
```

prints all feature sequences.

$gb->cds()

Returns the array of all cds object name.

!CAUTION! the object name is actually the FEATURE OBJECT NAME, to enable access to all feature values. However, most of the time you do not need to be aware of this difference.

```
foreach ($gb->cds()){
    $gb->get_geneseq($_);
}
```

prints all gene sequences.

$gb->startcodon()

Given a CDS ID, returns the start codon.

eg. $gb->startcodon('CDS1'); returns 'atg'

$gb->stopcodon()

Given a CDS ID, returns the stop codon.

eg. $gb->stopcodon('CDS1'); returns 'tag'

$gb->before_startcodon()

Given a CDS ID and length, returns the sequence upstream of start codon.

eg. $gb->before_startcodon('CDS1', 100); returns 100 bp sequence upstream of the start codon of 'CDS1'.

$gb->after_startcodon()

Given a CDS ID and length, returns the sequence downstream of start codon.

eg. $gb->after_startcodon('CDS1', 100); returns 100 bp sequence downstream of the start codon of 'CDS1'.

$gb->before_stopcodon()

Given a CDS ID and length, returns the sequence upstream of stop codon.

eg. $gb->before_stopcodon('CDS1', 100); returns 100 bp sequence upstream of the stop codon of 'CDS1'.

$gb->after_stopcodon()

Given a CDS ID and length, returns the sequence downstream of stop codon.

eg. $gb->after_stopcodon('CDS1', 100); returns 100 bp sequence downstream of the stop codon of 'CDS1'.

$gb->get_intron()

Given a CDS ID, returns the intron sequences as array of sequences.

eg. $gb->get_intron('CDS1');

returns ($1st_intron, $2nd_intron,..)

$gb->pos2feature()

Given a GenBank position (sequence starting from position 1) returns the G-instance ID (ex. FEATURE123) of the feature at the given position. If multiple features exists for the given position, the first feature to appear is returned. Returns NULL if no feature exists.

$gb->pos2gene()

Given a GenBank position (sequence starting from position 1) returns the G-instance ID (ex. FEATURE123) of the gene at the given position. If multiple genes exists for the given position, the first gene to appear is returned. Returns NULL if no gene exists.

$gb->gene2id()

Given a GenBank gene name, returns the G object feature ID (ex. FEATURE123). Returns NULL if no gene exists.

$gb->get_exon()

Given a CDS ID, returns the exon sequence.

'complement' is properly parsed, and introns are spliced out.

eg. $gb->get_exon('CDS1'); returns the 'CDS1' exon.

$gb->next_locus()

Reads the next locus.

the G instance is then updated.

```
do{

}while($gb->next_locus());
```

Enables multiple loci analysis.

$gb->next_seq()

If G instance is created with 'long sequence' option,
$gb->next_seq() method replace the next chunk of sequence
to $gb->{SEQ}.

```
while($gb->next_seq(100000)){
    print $gb->{SEQ};
}
```

Enables continuous analysis.

$gb->rewind_genome()

If G instance is created with 'long sequence' option,
$gb->rewind_genome() method puts the filehandle pointer back
to the ORIGIN position.