Sequence Analysis with R and Bioconductor
Overview

Thomas Girke

December 11, 2011
Overview

String Handling Utilities in R’s Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
Outline

Overview

String Handling Utilities in R’s Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
Biosequence Analysis in R and Bioconductor

R Base

- Some basic string handling utilities. Wide spectrum of numeric data analysis tools.

Bioconductor

- Bioconductor packages provide much more sophisticated string handling utilities for sequence analysis.
  - Biostrings: general sequence analysis environment
  - ShortRead: pipeline for short read data
  - IRanges: low-level infrastructure for range data
  - GenomicRanges: high-level infrastructure for range data
  - BSgenome: genome annotation data
  - biomaRt: interface to BioMart annotations
  - rtracklayer: Annotation imports, interface to online genome browsers

Interface for non-R sequence analysis tools

- e.g. short read aligners
Overview

String Handling Utilities in R's Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
Basic String Matching and Parsing

String matching.

```r
> myseq <- c("ATGCAGACATAGTG", "ATGAACATAGATCC", "GTACAGATCAC") # Sample sequence data set.
> myseq[grep("ATG", myseq)] # String searching with regular expression support.
[1] "ATGCAGACATAGTG" "ATGAACATAGATCC"

> pos1 <- regexpr("AT", myseq) # Searches 'myseq' for first match of pattern "AT".
> as.numeric(pos1); attributes(pos1)$match.length # Returns position information of matches.
[1] 1 1 7
[1] 2 2 2

> pos2 <- gregexpr("AT", myseq) # Searches 'myseq' for all matches of pattern "AT".
> as.numeric(pos2[[1]]); attributes(pos2[[1]])$match.length # Returns positions of matches in first sequence.
[1] 1 9
[1] 2 2

> gsub("^ATG", "atg", myseq) # String substitution with regular expression support.
[1] "atgCAGACATAGTG" "atgAACATAGATCC" "GTACAGATCAC"
```

Positional parsing.

```r
> nchar(myseq) # Computes length of strings.
[1] 14 14 11

> substring(myseq[1], c(1,3), c(2,5)) # Positional parsing of several fragments from one string.
[1] "AT" "GCA"

> substring(myseq, c(1,4,7), c(2,6,10)) # Positional parsing of many strings.
[1] "AT" "AAC" "ATCA"
```
Random Sequence Generation

Create any number of random DNA sequences of any length.

```r
> rand <- sapply(1:100, function(x) paste(sample(c("A","T","G","C"), sample(10:20), replace=T), collapse=""))
> rand[1:3]

[1] "TTATATAGCT"     "TTAAGATGTCCCTCGGG"  "ACGACCATGAAC"
```

Enumerate sequences to check for duplicates.

```r
> table(c(rand[1:4], rand[1]))

     ACGACCATGAAC CCGATGCAAGCCATCGGGA TTAAGATGTCCCTCGGG TTATATATAGCT
   [1,]      1          1        1          2
```

Extract any number of pseudo reads from the following reference. Note: this requires Biostrings.

```r
> library(Biostrings)
> ref <- DNAString(paste(sample(c("A","T","G","C"), 100000, replace=T), collapse=""))
> randstart <- sample(1:(length(ref)-15), 1000)
> randreads <- Views(ref, randstart, width=15)
> rand_set <- DNAStringSet(randreads)
> unlist(rand_set)

15000-letter "DNAString" instance
seq: TAATAGGCCCCTCCTAGTCTCTAAGTGTGATGATGATGCTGGGTAAGAAGCGCCCTCGATGTAGTACCTAGGCCATTAAGAGACGACACGACAGCTACG
```
Outline

Overview

String Handling Utilities in R’s Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
Important Data Objects in Biostrings

**XString for single sequence**
- **DNAString**: for DNA
- **RNAString**: for RNA
- **AAString**: for amino acid
- **BString**: for any string

**XStringSet for many sequences**
- **DNAStringSet**: for DNA
- **RNAStringSet**: for RNA
- **AAStringSet**: for amino acid
- **BStringSet**: for any string

**QualityScaleXStringSet for many sequences plus quality data**
- **QualityScaledDNAStringSet**: for DNA
- **QualityScaledRNAStringSet**: for RNA
- **QualityScaledAAStringSet**: for amino acid
- **QualityScaledBStringSet**: for any string
Download these following sequences to current working directory and then import them into R:

```r
> myseq <- read.DNAStringSet("AE004437.ffn")
> myseq[1:3]

A DNAStringSet instance of length 3
width seq
[1] 1206 ATGACTCGCGGTCTCGTGTCGGTGCCGGCCTCGCAGCCATTGTACTGGCCCTGGCCGCAGTGTCGGCTGCCGCTCCGATTGCCGGGGCGCAG...AGCGGTGGCGGGTTACCGCTGTTCAAGATCGGGGGCGCTGTCGCTGTGATTGCGATCGTCGTCGTCGTTGTTCGACGCTGGCGGAACCCATGA gb|AE004437.1|:24...
[2]  666 ATGAGCATCATACTCGAAGGCGTGGTCAAACGGTACGAAACCGGTGCCGAGACAGTCGAGGCGCTGAAAGGCGTTGACTTCTCGGCGGCG...AACATCGCCGTTGGTTGCGATCACTCACGACACGCAACTCGAGGAGTTCTCCGACCGCGCAGTCAACCTCGTCGATGGGGTGTTACACACGTGA gb|AE004437.1|:14...
[3] 1110 ATGGCGTGGCGGAACCTCGGGCGGAACCGCGTGCGGACTGCGCTGGCCGCGCTCGGGATCGTGATCGGTGTGATCTCGATCGCATCGATGGGG...TTCCTGTTCGCGGTCTTCGCCAGCCTGCTCAGCGGGCTCTATCCGGCGTGGAAAGCAGCCAACGATCCGCCCGTCGAGGCGCTCGGCGAATGA gb|AE004437.1|:21...
```

```r
> sub <- myseq[grep("99.*", names(myseq))]
> length(sub)

[1] 185
```

```r
> write.XStringSet(sub, file="AE004437sub.ffn", width=80)

Open exported sequence file AE004437sub.ffn in a text editor.
The XString stores the different types of biosequences in dedicated containers:

```r
> library(Biostrings)
> d <- DNAString("GCATAT-TAC")
> d

10-letter "DNAString" instance
seq: GCATAT-TAC

> d[1:4]

4-letter "DNAString" instance
seq: GCAT

> r <- RNAString("GCAUAU-UAC")
> r <- RNAString(d) # Converts d into RNAString object.
> p <- AAString("HCWYHH")
> b <- BString("I store any set of characters. Other XString objects store only the IUPAC characters.")
```
XStringSet containers allow to store many biosequences in one object:

```r
> dset <- DNAStringSet(c("GCATATTAC", "AATCGATCC", "GCATATTAC"))
> names(dset) <- c("seq1", "seq2", "seq3") # Assigns names
> dset[1:2]

A DNAStringSet instance of length 2
  width  seq  names
[1] 9  GCATATTAC
[2] 9  AATCGATCC

> width(dset) # Returns the length of each sequence
[1] 9 9 9

> d <- dset[[1]] # The [[ subsetting operator returns a single entry as XString object
> dset2 <- c(dset, dset) # Appends/concatenates two XStringSet objects
> dsetchar <- as.character(dset) # Converts XStringSet to named vector
> dsetone <- unlist(dset) # Collapses many sequences to a single one stored in a DNAString container

Sequence subsetting by positions:

> DNAStringSet(dset, start=c(1,2,3), end=c(4,8,5))

A DNAStringSet instance of length 3
  width  seq
[1] 4 GCAT
[2] 7 ATCGATC
[3] 3 ATA
The XMultipleAlignment class stores the different types of multiple sequence alignments:

```r
> origMAlign <- read.DNAMultipleAlignment(filepath = system.file("extdata",
    "msx2_mRNA.aln", package = "Biostrings"), format = "clustal")
> origMAlign
```

DNAMultipleAlignment with 8 rows and 2343 columns

```
aln
[1] -----TCCGTCTCCGCAGCAAAAAAAGTTTTGAGTCGCCGCTCCCGGGTTGCCACGCGAGTCGCGCGTCGGGAGCTACGTAGGGCAGAGAAGTCA-T...GAAGAGTTATCTCTTATTCTGAATT--AAATTAAGC--ATTTGTTTTATTGCAGTAAAGTTTGTCCAAACTCACAATTAAAAAAAAAAAAAAAAA
gi|84452153|ref|

gi|208431713|ref|

gi|118601823|ref|

gi|114326503|ref|

gi|119220589|ref|

gi|148540149|ref|

[7] --------------CGGCTCCGCAGCGCCTCACTCGCGCAGTCCCCCGCGCAGGGCCGGGCAGAGGCGCACGCAGCTCCCCGGCGGCCCCGCTC-C...-----------------------------------------------------------------------------------------------
gi|45383056|ref|

[8] GGGGGAGACTTCAGAAGTTGTGGTCTTCGCTCCTCCGCTGATAAACAGTTGAGATGCACATATTATTATTACTTTTTAGGACAAGTTGAATGTGTTCGTAAC...---
```
Phred quality scores are integers from 0-50 that are stored as ASCII characters after adding 33. The basic R functions rawToChar and charToRaw can be used to interconvert among their representations.

```r
> phred <- 1:9
> phreda <- paste(sapply(as.raw((phred)+33), rawToChar), collapse=""); phreda

[1] ""#$%&'()*+,-./0123456789:

> as.integer(charToRaw(phreda))-33

[1] 1 2 3 4 5 6 7 8 9
```

```r
> dset <- DNAStringSet(sapply(1:100, function(x) paste(sample(c("A","T","G","C"), 20, replace=T), collapse=""))); dset
> myqlist <- lapply(1:100, function(x) sample(1:40, 20, replace=T)) # Creates random Phred score list.
> myqual <- sapply(myqlist, function(x) toString(PhredQuality(x))) # Converts integer scores into ASCII char.
> myqual <- PhredQuality(myqual) # Converts to a PhredQuality object.
> dsetq1 <- QualityScaledDNAStringSet(dset, myqual) # Combines DNAStringSet and quality data in QualityScaledDNAStringSet object.
> dsetq1[1:2]

A QualityScaledDNAStringSet instance containing:
A DNAStringSet instance of length 2
  width  seq
[1] 20 AGACTGGACTCATGGAATGG
[2] 20 AGTAGTATCAAGTATTCACC

A PhredQuality instance of length 2
  width  seq
[1] 20 CB?2<71+#749CG#7D+<>
[2] 20 I'37299)1*C)F7'Q.B)
```

See ShortRead for additional utilities!
Basic Sequence Manipulations

Complement, reverse, and reverse & complement of sequences:

```r
> randset <- DNAStringSet(rand)
> complement(randset[1:2])

A DNAStringSet instance of length 2
  width seq
[1]  12 AATATATATCGA
[2]  17 AATTCTACAGGGAGCCC

> reverse(randset[1:2])

A DNAStringSet instance of length 2
  width seq
[1]  12 TCGATATATATT
[2]  17 GGGCTCCCTGTAGAATT

> reverseComplement(randset[1:2])

A DNAStringSet instance of length 2
  width seq
[1]  12 AGCTATATATAA
[2]  17 CCCGAGGGACATCTTAA

Translate DNA sequences into proteins:

```r
> translate(randset[1:2])

A AAStringSet instance of length 2
  width seq
[1]   4 LYIA
[2]   5 LRCPS
Pattern Matching

Pattern matching with mismatches

```r
> mypos <- matchPattern("ATGGTG", myseq1[[1]], max.mismatch=1) # Finds pattern matches in reference
> countPattern("ATGGCT", myseq1[[1]], max.mismatch=1) # Counts only the corresponding matches
> tmp <- c(DNAStringSet("ATGGTG"), DNAStringSet(mypos)) # Results shown in DNAStringSet object
> consensusMatrix(tmp) # Returns a consensus matrix for query and hits.
> myvpos <- vmatchPattern("ATGGCT", myseq1, max.mismatch=1) # Finds all pattern matches in reference
> myvpos # The results are stored as MIndex object.
> Views(myseq1[[1]], start(myvpos[[1]]), end(myvpos[[1]])) # Retrieves the result for single entry
> sapply(seq(along=myseq1), function(x) as.character(Views(myseq1[[x]], start(myvpos[[x]]), end(myvpos[[x]])))) # All matches.
```

Pattern matching with regular expression support

```r
> myseq <- DNAStringSet(c("ATGCAGACATAGTG", "ATGAACATAGATCC", "GTACAGATCAC"))
> myseq[grep("^ATG", myseq, perl=TRUE)] # String searching with regular expression support
> pos1 <- regexpr("AT", myseq) # Searches 'myseq' for first match of pattern "AT"
> as.numeric(pos1); attributes(pos1)$match.length # Returns position information of matches
> pos2 <- gregexpr("AT", myseq) # Searches 'myseq' for all matches of pattern "AT"
> as.numeric(pos2[[1]]); attributes(pos2[[1]])$match.length # Match positions in first sequence
> DNAStringSet(gsub("^ATG", "NNN", myseq)) # String substitution with regular expression support
```
> pwm <- PWM(DNAStringSet(c("GCT", "GGT", "GCA")))
> library(seqLogo); seqLogo(t(t(pwm) * 1/colSums(pwm)))

Views on a 21-letter DNAString subject
subject: AAAGCTAAAGGTAAAGCAAAA
views:
  start   end width
[1] 4     6    3 [GCT]
[2] 10   12    3 [GGT]
[3] 16   18    3 [GCA]
Outline

Overview

String Handling Utilities in R’s Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
**Important Data Objects for Range Operations**

- **IRanges**: stores range data only (IRanges library)
- **GRanges**: stores ranges and annotations (GenomicRanges library)
- **GRangesList**: list version of GRanges container (GenomicRanges library)
Range Data are Stored in IRanges and GRanges Containers

### Constructing GRanges Objects

```r
> library(GenomicRanges); library(rtracklayer)
> gr <- GRanges(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)), ranges = IRanges(1:10, end = 7:16, ...) 3, 2)), score = 1:10, GC = seq(1, 0, length = 10)) # Example of creating a GRanges object with its constructor function.
> seqlengths(gff) <- end(ranges(gff[which(elementMetadata(gff)[,"type"]=='chromosome")],]))
> names(gff) <- 1:length(gff) # Assigns names to corresponding slot.
> gff[1:4,
GRanges with 4 ranges and 4 elementMetadata values:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>type</th>
<th>source</th>
<th>phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;factor&gt;</td>
<td>&lt;factor&gt;</td>
<td>&lt;factor&gt;</td>
</tr>
<tr>
<td>1 Chr1</td>
<td>[ 1, 30427671]</td>
<td>+</td>
<td>chromosome</td>
<td>TAIR10</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>2 Chr1</td>
<td>[3631, 5899]</td>
<td>+</td>
<td>gene</td>
<td>TAIR10</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>3 Chr1</td>
<td>[3631, 5899]</td>
<td>+</td>
<td>mRNA</td>
<td>TAIR10</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>4 Chr1</td>
<td>[3760, 5630]</td>
<td>+</td>
<td>protein</td>
<td>TAIR10</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>

seqlengths:

<table>
<thead>
<tr>
<th>Chr1</th>
<th>Chr2</th>
<th>Chr3</th>
<th>Chr4</th>
<th>Chr5</th>
<th>ChrC</th>
<th>ChrM</th>
</tr>
</thead>
<tbody>
<tr>
<td>30427671</td>
<td>19698289</td>
<td>23459830</td>
<td>18585056</td>
<td>26975502</td>
<td>154478</td>
<td>366924</td>
</tr>
</tbody>
</table>

> gff_rd <- as(gff, "RangedData") # Coerces GRanges object to RangedData class.
> gff_gr <- as(gff_rd, "GRanges") # Coerces RangedData object to GRanges class.
```
Utilities for Range Containers

Accessor and subsetting methods for GRanges objects

> c(gff[1:2], gff[401:402]) # GRanges objects can be concatenated with the c() function.
> seqnames(gff); ranges(gff); strand(gff); seqlengths(gff) # Accessor functions
> start(gff[1:4]); end(gff[1:4]); width(gff[1:4]) # Direct access to IRanges components
> elementMetadata(gff); elementMetadata(gff)[, "type"] # Accessing metadata component.
> gff[elementMetadata(gff)[, "type"] == "gene"] # Returns only gene ranges.

Useful utilities for GRanges objects

> strand(gff) <- "*" # Erases the strand information
> reduce(gff) # Collapses overlapping ranges to continuous ranges.
> gaps(gff) # Returns uncovered regions.
> disjoin(gff) # Returns disjoint ranges.
> coverage(gff) # Returns coverage of ranges.
> findOverlaps(gff, gff[1:4]) # Returns the index pairings for the overlapping ranges.
> countOverlaps(gff, gff[1:4]) # Counts overlapping ranges
> subsetByOverlaps(gff, gff[1:4]) # Returns only overlapping ranges

GRangesList Objects

> sp <- split(gff) # Stores every range in separate component of a GRangesList object
> split(gff, seqnames(gff)) # Stores ranges of each chromosome in separate component.
> unlist(sp) # Returns data as GRanges object
> sp[1:4, "type"] # Subsetting of GRangesList objects is similar to GRanges objects.
> lapply(sp[1:4], length); sapply(sp[1:4], length) # Looping over GRangesList objects similar to lists
Outline

Overview

String Handling Utilities in R’s Base Distribution

Sequence Handling with Bioconductor

Range Operations

Exercises
Exercise

GFF from *Halobacterium sp* [Link](#)

Genome from *Halobacterium sp* [Link](#)

**Task 1** Extract gene ranges, parse their sequences from genome and translate them into proteins

**Task 2** Reduce overlapping genes and parse their sequences from genome

**Task 3** Generate intergenic ranges and parse their sequences from genome

**Useful commands**

```r
> chr <- read.DNAStringSet("AE004437.fna")
> writeLines(readLines("AE004437.gff")[-c(1:7)], "AE004437.gff2")
> gff <- import.gff("AE004437.gff2", asRangedData=FALSE)
> gffgene <- gff[elementMetadata(gff)[,"type"]=='gene']
> gene <- DNAStringSet(Views(chr[[1]], IRanges(start(gffgene), end(gffgene))))
> names(gene) <- elementMetadata(gffgene)[,"group"]
> pos <- elementMetadata(gffgene[strand(gffgene) == "+"])[,"group"]
> translate(gene[names(gene) %in% pos])
> neg <- elementMetadata(gffgene[strand(gffgene) == "]-" ])[,"group"]
> translate(reverseComplement(gene[names(gene) %in% neg]))
```