Chapter 31: Run-length encoding

Section 31.1: Run-length Encoding with `rle`

Run-length encoding captures the lengths of runs of consecutive elements in a vector. Consider an example vector:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

The `rle` function extracts each run and its length:

```
r <- rle(dat)
r
# Run Length Encoding
#   lengths: int [1:6] 1 3 1 1 2 2
#   values : num [1:6] 1 2 3 1 4 1
```

The values for each run are captured in `r$values`:

```
r$values
# [1] 1 2 3 1 4 1
```

This captures that we first saw a run of 1's, then a run of 2's, then a run of 3's, then a run of 1's, and so on.

The lengths of each run are captured in `r$lengths`:

```
r$lengths
# [1] 1 3 1 1 2 2
```

We see that the initial run of 1's was of length 1, the run of 2's that followed was of length 3, and so on.

Section 31.2: Identifying and grouping by runs in base R

One might want to group their data by the runs of a variable and perform some sort of analysis. Consider the following simple dataset:

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1 1 1
# 2 1 2
# 3 2 3
# 4 2 4
# 5 2 5
# 6 1 6
```

The variable `x` has three runs: a run of length 2 with value 1, a run of length 3 with value 2, and a run of length 1 with value 1. We might want to compute the mean value of variable `y` in each of the runs of variable `x` (these mean values are 1.5, 4, and 6).

In base R, we would first compute the run-length encoding of the `x` variable using `rle`:

```
(r <- rle(dat$x))
# Run Length Encoding
#   lengths: int [1:3] 2 3 1
#   values : num [1:3] 1 2 1
```

The next step is to compute the run number of each row of our dataset. We know that the total number of runs is `length(r$lengths)`, and the length of each run is `r$lengths`, so we can compute the run number of each of our runs with `rep`:

```
(run.id <- rep(seq_along(r$lengths), r$lengths))
# [1] 1 1 2 2 2 3
```

Now we can use `tapply` to compute the mean y value for each run by grouping on the run id:

```
data.frame(x=r$values, meanY=tapply(dat$y, run.id, mean))
#   x meanY
# 1 1   1.5
# 2 2   4.0
# 3 1   6.0
```

Section 31.3: Run-length encoding to compress and decompress vectors

Long vectors with long runs of the same value can be significantly compressed by storing them in their run-length encoding (the value of each run and the number of times that value is repeated). As an example, consider a vector of length 10 million with a huge number of 1's and only a small number of 0's:

```
set.seed(144)
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)
table(dat)
#       0       1
#     103 9999897
```

Storing 10 million entries will require significant space, but we can instead create a data frame with the run-length encoding of this vector:

```
rle.df <- with(rle(dat), data.frame(values, lengths))
dim(rle.df)
# [1] 207   2
head(rle.df)
#   values lengths
# 1      1   52818
# 2      0       1
# 3      1  219329
# 4      0       1
# 5      1  318306
# 6      0       1
```

From the run-length encoding, we see that the first 52,818 values in the vector are 1's, followed by a single 0, followed by 219,329 consecutive 1's, followed by a 0, and so on. The run-length encoding only has 207 entries, requiring us to store only 414 values instead of 10 million values. As `rle.df` is a data frame, it can be stored using standard functions like `write.csv`.

Decompressing a vector in run-length encoding can be accomplished in two ways. The first method is to simply call `rep`, passing the `values` element of the run-length encoding as the first argument and the `lengths` element of the run-length encoding as the second argument:

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

We can confirm that our decompressed data is identical to our original data:

```
identical(decompressed, dat)
# [1] TRUE
```

The second method is to use R's built-in `inverse.rle` function on the `rle` object, for instance:

```
rle.obj <- rle(dat)                          # create a rle object here
class(rle.obj)
# [1] "rle"

dat.inv <- inverse.rle(rle.obj)              # apply the inverse.rle on the rle object
```

We can confirm again that this produces exactly the original `dat`:

```
identical(dat.inv, dat)
# [1] TRUE
```

Section 31.4: Identifying and grouping by runs in data.table

The data.table package provides a convenient way to group by runs in data. Consider the following example data:

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#    x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

The variable `x` has three runs: a run of length 2 with value 1, a run of length 3 with value 2, and a run of length 1 with value 1. We might want to compute the mean value of variable `y` in each of the runs of variable x (these mean values are 1.5, 4, and 6).

The data.table `rleid` function provides an id indicating the run id of each element of a vector:

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

One can then easily group on this run ID and summarize the `y` data:

```
DT[,mean(y),by=.(x, rleid(x))]
#    x rleid  V1
# 1: 1     1 1.5
# 2: 2     2 4.0
# 3: 1     3 6.0
```