CMSC858D Homework 1 : Implementing some succinct primitives

Jayaram Kancherla

10/27/2019

Building the package

The code is written in `go` (version go1.10.4 linux/amd64). To build the package

```
cd wt
go build
```

To print the available operations within the tool, run

```
wt help
```

The benchmarks were run on a 64 bit machine running ubuntu using the WSL. To run the benchmarks locally

```
wt runtests
```

Load packages

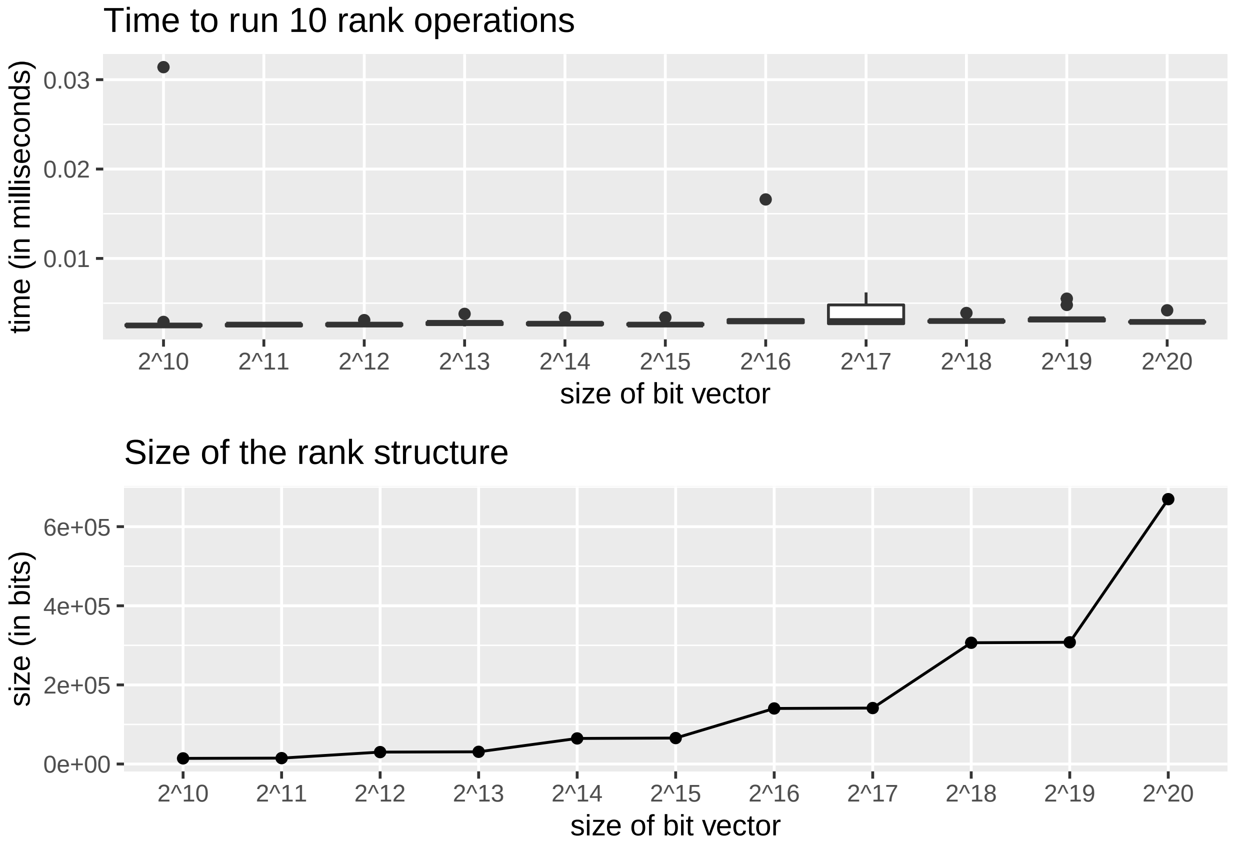
```r
library(rjson)
library(ggplot2)
library(gridExtra)
library(reshape2)
```

Task 1 — bit-vector rank

For this programming task, test your implementation by invoking it for bit vectors of various sizes, and plotting the bit-vector size (say N) versus the time requried to do some fixed number of rank operations. Also, plot the bit-vector size (say N) versus the result of calling the overhead() function. Does your implementation match the expected theoretical bounds?

```r
times <- rjson::fromJSON(file="wt/task1-times.json")
xlabels <- paste0("2^", seq(from=10, to=20))
names(times) <- xlabels
times <- as.data.frame(times)
colnames(times) <- xlabels
times$run <- paste0("run", seq(1,11))

dmelt <- melt(times, id.vars = c("run"))

plot1 <- ggplot(data=dmelt, aes(x = variable, y=value/1000000)) +
  geom_boxplot() +
  xlab("size of bit vector") +
  ylab("time (in milliseconds)") +
  labs(title="Time to run 10 rank operations")
# plot1

sizes <- rjson::fromJSON(file="wt/task1-size.json")
names(sizes) <- xlabels
```

```r
data <- data.frame(sizes = sizes, xlabels = xlabels)

plot2 <- ggplot(data = data, aes(x = xlabels, y=sizes, group=1)) +
  geom_line() + geom_point() +
  xlab("size of bit vector") +
  ylab("size (in bits)") +
  labs(title="Size of the rank structure")
# plot2

grid.arrange(plot1, plot2, ncol=1)
```

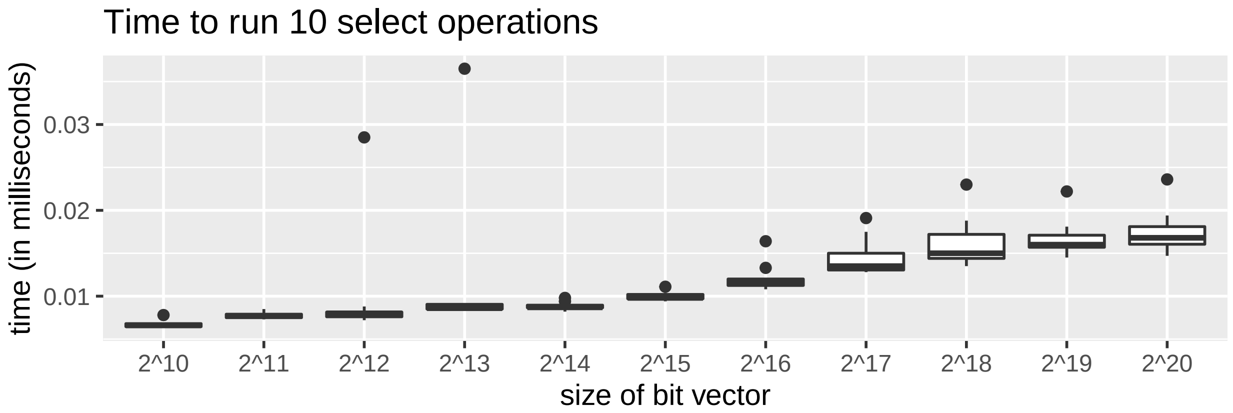


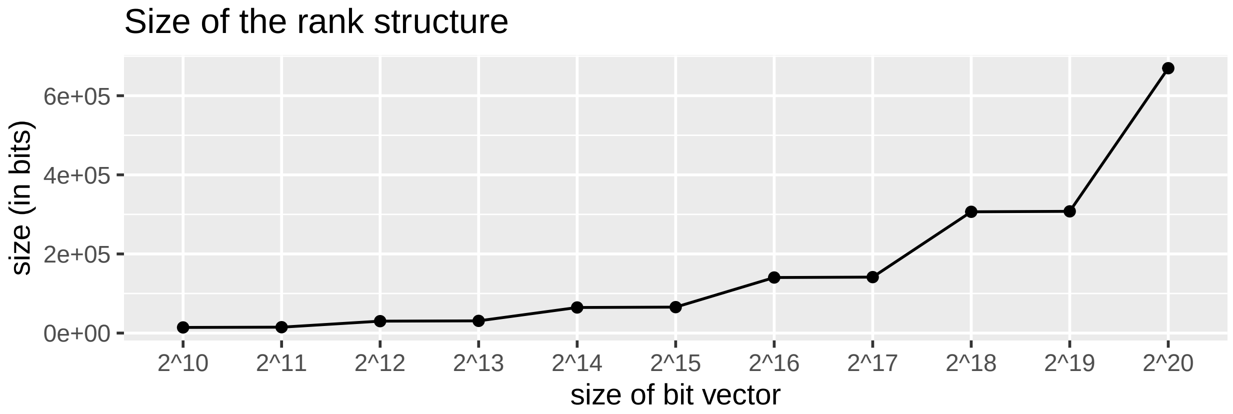
The tests were conducted by exponentially increasing the size of the bit vector (starting from 2^{10} to 2^{20}). For each bit vector, I measure the size of overhead the `rank_select` data structure and the time to run `rank1` operations on randomly selected indices. For each measure, I calculate the mean and standard deviation across 10 runs. The plot on the top displays the time to run a fixed number (=10) of `rank1` operations and the bottom displays the size of the overhead as the size of the bit vector increases. The plots indicate the rank operations are almost contant time and is independe tof the size of the bit vector. The size of overhead is atmost the size of the bitvector (Z + o(Z)), thus matching the theoretical bounds of succint rank data structure.

Task 2 — bit-vector select

For this programming task, test your implementation by invoking it for bit vectors of various sizes, and plotting the bit-vector size (say N) versus the time requried to do some fixed number of select operations. Also, plot the bit-vector size (say N) versus the result of calling the overhead() function. Does your implementation match the expected theoretical bounds? If you feel ambitious, you can additionally implement a constant-time bit-vector select, though this is not required.

```r
times <- rjson::fromJSON(file="wt/task2-times.json")
xlabels <- paste0("2^", seq(from=10, to=20))
names(times) <- xlabels
times <- as.data.frame(times)
colnames(times) <- xlabels
times$run <- paste0("run", seq(1,11))

dmelt <- melt(times, id.vars = c("run"))

plot1 <- ggplot(data=dmelt, aes(x = variable, y=value/1000000)) +
  geom_boxplot() +
  xlab("size of bit vector") +
  ylab("time (in milliseconds)") +
  labs(title="Time to run 10 select operations")
# plot1

sizes <- rjson::fromJSON(file="wt/task2-size.json")
names(sizes) <- xlabels

data <- data.frame(sizes = sizes, xlabels = xlabels)

plot2 <- ggplot(data = data, aes(x = xlabels, y=sizes, group=1)) +
  geom_line() + geom_point() +
  xlab("size of bit vector") +
  ylab("size (in bits)") +
  labs(title="Size of the rank structure")
# plot2

grid.arrange(plot1, plot2, ncol=1)
```





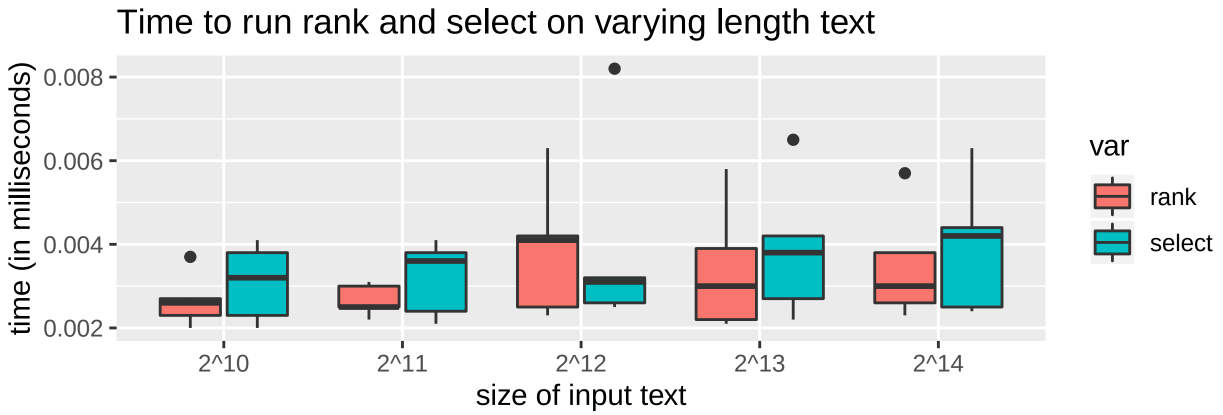
The tests were conducted exactly similar to the `rank` operations for Task 1. Instead of the rank, we now measure the time for 10 `select1` operations on randomly selected indices. Again the `select` operations are on the order of log compared to the constant time `rank` operations. Since the `select` operations use the same rank data structure, the size of the overhead did not change.

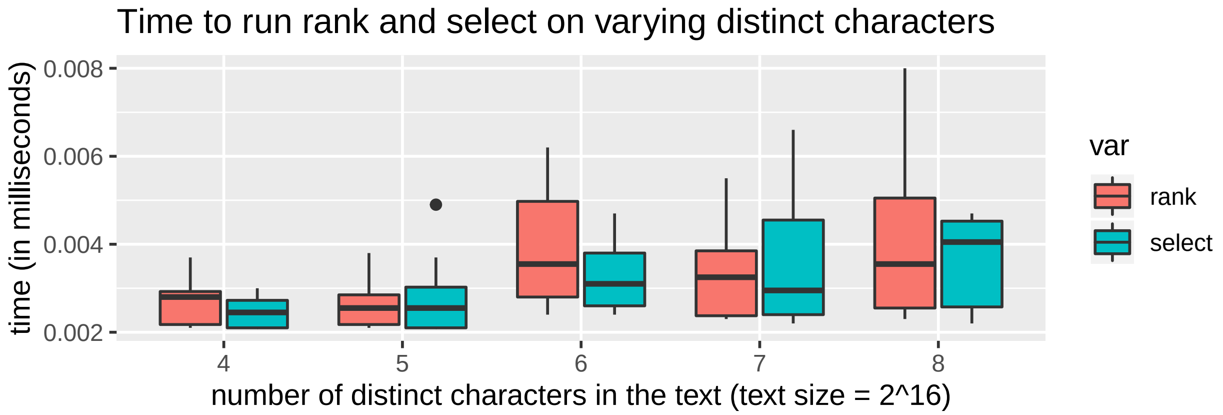
Task 3 — wavelet tree construction and query.

For this programming task, test your implementation by invoking it for some input strings of various sizes and with various numbers of distinct characters in the input string. Try plotting how (1) the rank and select queries in the wavelet tree scale as a factor of the string length and (2) how, at a fixed string length, how the rank and select queries scale as a factor of the number of distinct characters in the input string. How do your plots compare to expectation? Note: for the purpose of making these plots, you need not use the command-line interace described above. You can avoid the overhead by constructing the wavelet tree from the input, and then issuing the rank and select queries directly via function calls without serializing the tree to disk first.

```
times <- rjson::fromJSON(file="wt/task3.1-times.json")
xlabels <- paste0("2^", seq(from=10, to=14))
names(times) <- xlabels
times <- as.data.frame(times)
times <- t(times)
rownames(times) <- xlabels
colnames(times) <- paste0(rep(c("rank", "sele"), 5), seq(1,10))
times <- as.data.frame(times)
times$xlabels <- xlabels

dmelt <- melt(times, id.vars = c("xlabels"))
```

```r
dmelt$var <- substr(dmelt$variable, 1, 4)
dmelt$var[dmelt$var == "sele"] <- "select"

plot1 <- ggplot(data=dmelt, aes(x = xlabels, y=value/1000000, fill = var)) +
  geom_boxplot() +
  xlab("size of input text") +
  ylab("time (in milliseconds)") +
  labs(title="Time to run rank and select on varying length text ")
# plot1

times <- rjson::fromJSON(file="wt/task3.2-times.json")
times <- times[-c(1,2,3)]
xlabels <- seq(4, 8)
names(times) <- xlabels
times <- as.data.frame(times)
times <- t(times)
rownames(times) <- xlabels
colnames(times) <- paste0(rep(c("rank", "sele"), 8), seq(1,16))
times <- as.data.frame(times)
times$xlabels <- paste0(xlabels, "")

dmelt <- melt(times, id.vars = c("xlabels"))
dmelt$var <- substr(dmelt$variable, 1, 4)
dmelt$var[dmelt$var == "sele"] <- "select"

plot2 <- ggplot(data=dmelt, aes(x = xlabels, y=value/1000000, fill = var)) +
  geom_boxplot() +
  xlab("number of distinct characters in the text (text size = 2^16)") +
  ylab("time (in milliseconds)") +
  labs(title="Time to run rank and select on varying distinct characters")
# plot2

grid.arrange(plot1, plot2, ncol=1)
```





These plots visualize the time to run `rank` and `select` operations on Wavelet Trees. I run the operations across 5 different runs and calculate mean and standard deviation. The plot on the top measures the time to run these operations as the size of the text increases. The plot on the bottom measures the effect of alphabet size on these operations. The size of the text for this is fixed at 2^{16}. for the plot on the top, the `rank` & `select` operations increase on a log scale O(log k) matching the expectation times. For the plot on the bottom, the `rank` and `select` also seems to increase some log k (k = alphabet size) for varying the alphabet size.