CMSC858D Homework 1 : Implementing some succinct primitives

Jayaram Kancherla

11/25/2019

Source code available at https://github.com/jkanche/wavelet_trees

Building the package

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

```
# bloom filter
cd bf
go build

# blocked bloom filter
cd bbf
go build
```

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

```
bf runtests
bbf runtests
```

Load packages

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

```
## Loading required package: lattice
```

```
## Loading required package: plyr
```

Write Up

1. A short (no more than a paragraph) prose description of your implementation.

I am using the bit vector I implemented from the wavelet trees homework for this assignment. I've made minor changes to the implementation to get and set bits for an index. In addition the rest of the implementation is pretty straighforward, I create a new struct for the bloom filter data structure, given the number of unique keys and a false positive rate, I calculate the size of the bit vector and the number of hash functions to use. I use the murmur3 set of hash functions with different seeds (my search for finding different hash functions all said to use different seeds). I then set and read bits at different bit positions defined by the hash function.

2. A brief (again, no more than a paragraph) summary of what you found to be the most difficult part of implementing each task.

I think this programming assignment was pretty straight forward. I spent a good amount of time on how to find different hash functions but did not realize all i had to do was set different seeds!

Task 1 — Basic Bloom Filter

Writeup: For this programming task, test your implementation by creating Bloom filters for various key set sizes and various false positive rates (the exact parameters are up to you, but I suggest to vary N from at least a few thousand to a few million, and the fpr from ~1% to ~25%). For each combination of Bloom filter size and FPR, issue 3 sets of queries (1) a set of queries where no query key was present in the original set, (2) a set of queries where ~50% of the queries were present in the original set, and (3) a set of queries where all of the queries were present in the original set. For all of these query streams, report and plot the average query time for answering a query in your Bloom filter. For (1) and (2), also report and plot the empirical false positive rate of your Bloom filter (note, this doesn't make sense for (3) since there are no false positive keys, and so the FPR should be 0%).

```r
times_abs <- rjson::fromJSON(file="bf/abscenttimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_abs) <- xlabels
times_abs <- as.data.frame(times_abs)
colnames(times_abs) <- xlabels
times_abs$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_abs <- melt(times_abs, id.vars = c("run"))

dmelt_abs$value <- dmelt_abs$value / 1000000000
avg_times <- summarySE(dmelt_abs, measurevar="value", groupvars=c("run", "variable"))
```

```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

```r
plot1 <- ggplot(data=avg_times, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (Keys not present)")
# plot1

times_mix <- rjson::fromJSON(file="bf/mixedtimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_mix) <- xlabels
times_mix <- as.data.frame(times_mix)
colnames(times_mix) <- xlabels
times_mix$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_mix <- melt(times_mix, id.vars = c("run"))

dmelt_mix$value <- dmelt_mix$value / 1000000000
avg_times <- summarySE(dmelt_mix, measurevar="value", groupvars=c("run", "variable"))
```

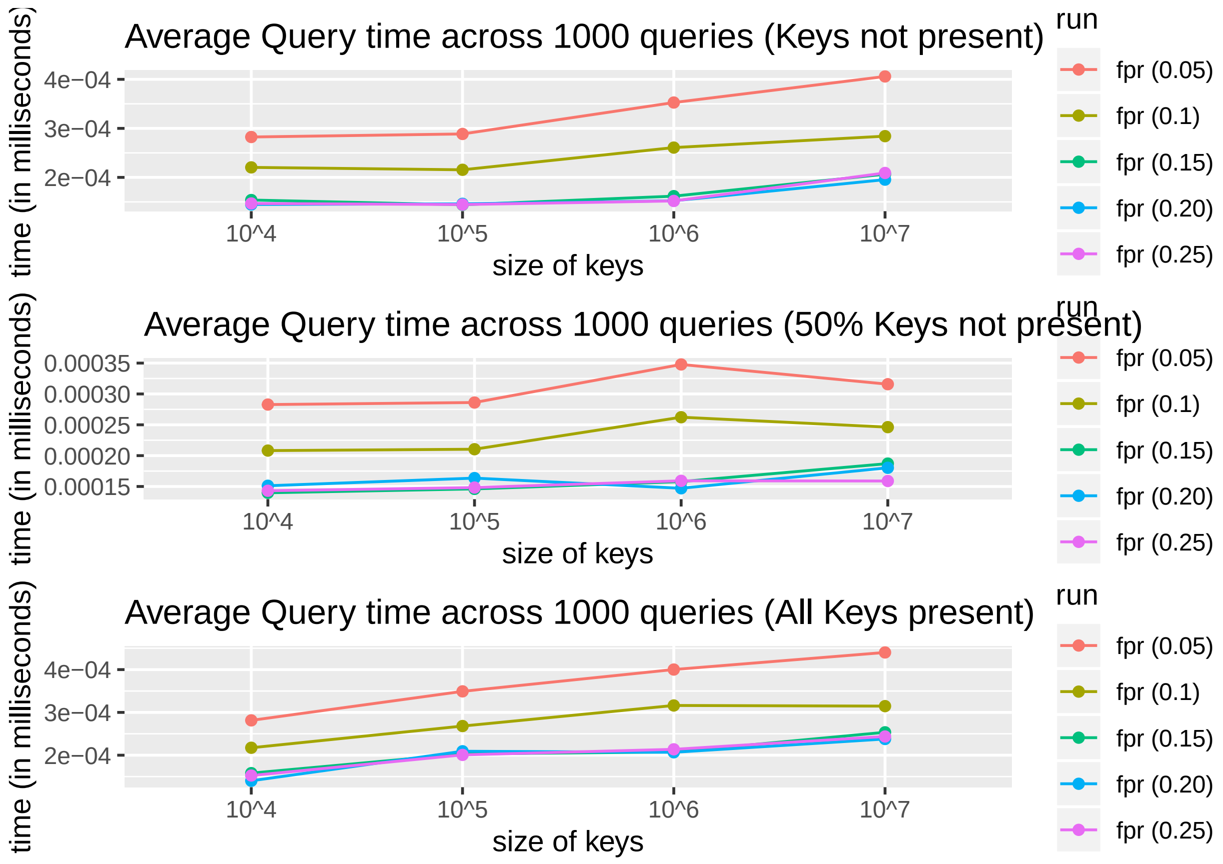
```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

```r
plot2 <- ggplot(data=avg_times, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (50% Keys not present)")
# plot2
```

```
times_pre <- rjson::fromJSON(file="bf/presenttimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_pre) <- xlabels
times_pre <- as.data.frame(times_pre)
colnames(times_pre) <- xlabels
times_pre$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_pre <- melt(times_pre, id.vars = c("run"))

# merged <- rbind(dmelt_abs, dmelt_mix, dmelt_pre)
dmelt_pre$value <- dmelt_pre$value / 1000000000
avg_times <- summarySE(dmelt_pre, measurevar="value", groupvars=c("run", "variable"))
```

```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

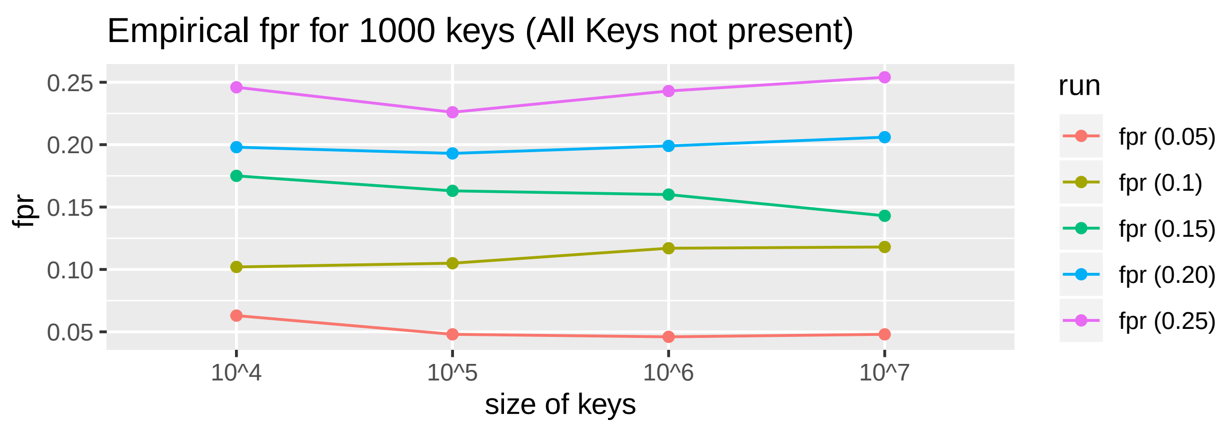
```
plot3 <- ggplot(data=avg_times, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (All Keys present)")
# plot3

grid.arrange(plot1, plot2, plot3)
```

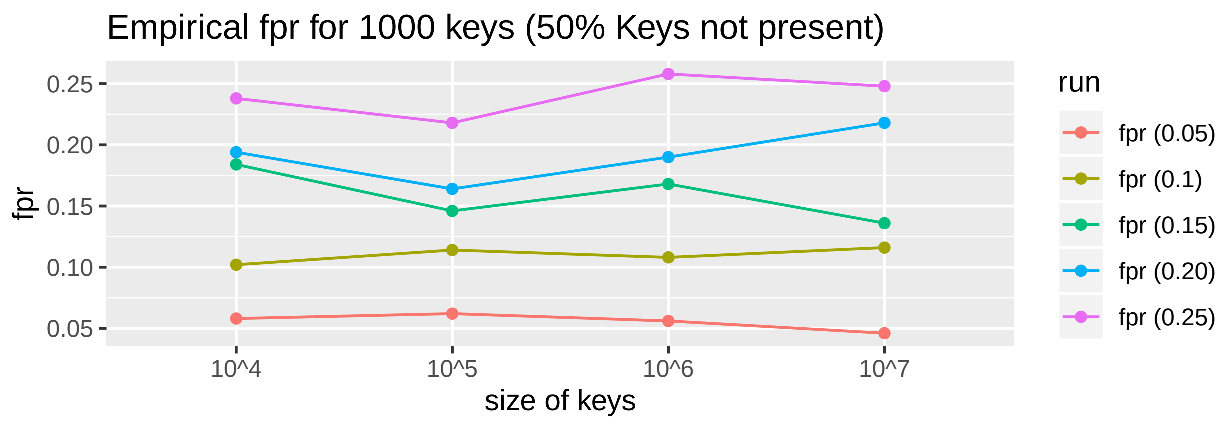


The plots shown here are the average query times for querying 1000 keys with the bloom filter. (top to bottom) 1) The first plot shows the average query time when no key in the search list is present in the bloom filter, 2) the second plot shows the average query time when 50% of the keys are present in the bloom filter and, 3) the third plot shows when all keys are present in the bloom filter. Irrespective of the search list, the average query time does not significantly change when the size of the bloom filter increases. it is independent of the size and the fpr.

False positive rate across 1000 queries

```
times_abs <- rjson::fromJSON(file="bf/absfpr.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_abs) <- xlabels
times_abs <- as.data.frame(times_abs)
colnames(times_abs) <- xlabels
times_abs$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_abs <- melt(times_abs, id.vars = c("run"))
dmelt_abs$value <- dmelt_abs$value / 1000

plot1 <- ggplot(data=dmelt_abs, aes(x=variable, y=value, group=run, color=run)) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("fpr ") +
  labs(title="Empirical fpr for 1000 keys (All Keys not present)")
# plot1

times_mix <- rjson::fromJSON(file="bf/mixfpr.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_mix) <- xlabels
times_mix <- as.data.frame(times_mix)
colnames(times_mix) <- xlabels
times_mix$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_mix <- melt(times_mix, id.vars = c("run"))
dmelt_mix$value <- dmelt_mix$value / 500

plot2 <- ggplot(data=dmelt_mix, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("fpr") +
  labs(title="Empirical fpr for 1000 keys (50% Keys not present)")
# plot2

grid.arrange(plot1, plot2)
```





These plots show the empirical fpr for two different scenarios, 1) top, where no keys from the search list are present in the bloom filter and 2) bottom where only 50% of the keys are present in the bloom filter. We notice that the fpr is within a margin of error close to the theoretical fpr. In the bottom plot, we show the empirical fpr when 50% of the keys are present in the bloom filter. Even in this scenario, our fpr is within the theoretical bounds.

Task 2 - Blocked Bloom Filter

Writeup: For this programming task, test your implementation by creating blocked Bloom filters for various key set sizes and various false positive rates (the exact parameters are up to you, but I suggest to vary N from at least a few thousand to a few million, and the fpr from ~1% to ~25%). For each combination of Bloom filter size and FPR, issue 3 sets of queries (1) a set of queries where no query key was present in the original set, (2) a set of queries where ~50% of the queries were present in the original set, and (3) a set of queries where all of the queries were present in the original set. For all of these query streams, report and plot the average query time for answering a query in your Bloom filter. For (1) and (2), also report and plot the empirical false positive rate of your Bloom filter (note, this doesn't make sense for (3) since there are no false positive keys, and so the FPR should be 0%). Pay attention to how the empirical FPR of the blocked Bloom filter compares to the empirical FPR of the standard Bloom filter, based upon the nominal FPR you request.

```
times_abs <- rjson::fromJSON(file="bbf/abscenttimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_abs) <- xlabels
times_abs <- as.data.frame(times_abs)
colnames(times_abs) <- xlabels
times_abs$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_abs <- melt(times_abs, id.vars = c("run"))
```

```r
dmelt_abs$value <- dmelt_abs$value / 1000000000
avg_times <- summarySE(dmelt_abs, measurevar="value", groupvars=c("run", "variable"))
```

```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

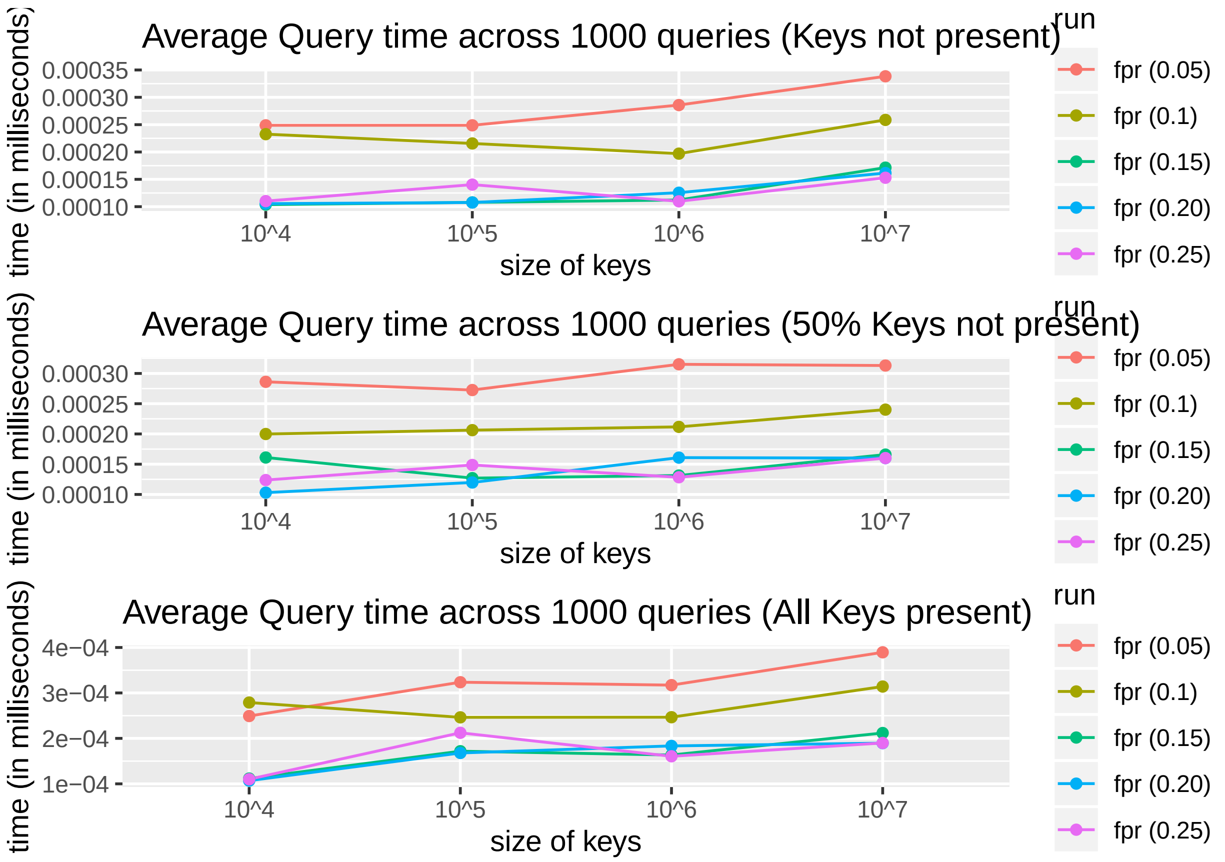
```r
plot1 <- ggplot(data=dmelt_abs, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (Keys not present)")
# plot1

times_mix <- rjson::fromJSON(file="bbf/mixedtimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_mix) <- xlabels
times_mix <- as.data.frame(times_mix)
colnames(times_mix) <- xlabels
times_mix$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_mix <- melt(times_mix, id.vars = c("run"))

dmelt_mix$value <- dmelt_mix$value / 1000000000
avg_times <- summarySE(dmelt_mix, measurevar="value", groupvars=c("run", "variable"))
```

```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

```r
plot2 <- ggplot(data=dmelt_mix, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (50% Keys not present)")
# plot2

times_pre <- rjson::fromJSON(file="bbf/presenttimes.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_pre) <- xlabels
times_pre <- as.data.frame(times_pre)
colnames(times_pre) <- xlabels
times_pre$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_pre <- melt(times_pre, id.vars = c("run"))

# merged <- rbind(dmelt_abs, dmelt_mix, dmelt_pre)
dmelt_pre$value <- dmelt_pre$value / 1000000000
avg_times <- summarySE(dmelt_pre, measurevar="value", groupvars=c("run", "variable"))
```

```
## Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced
```

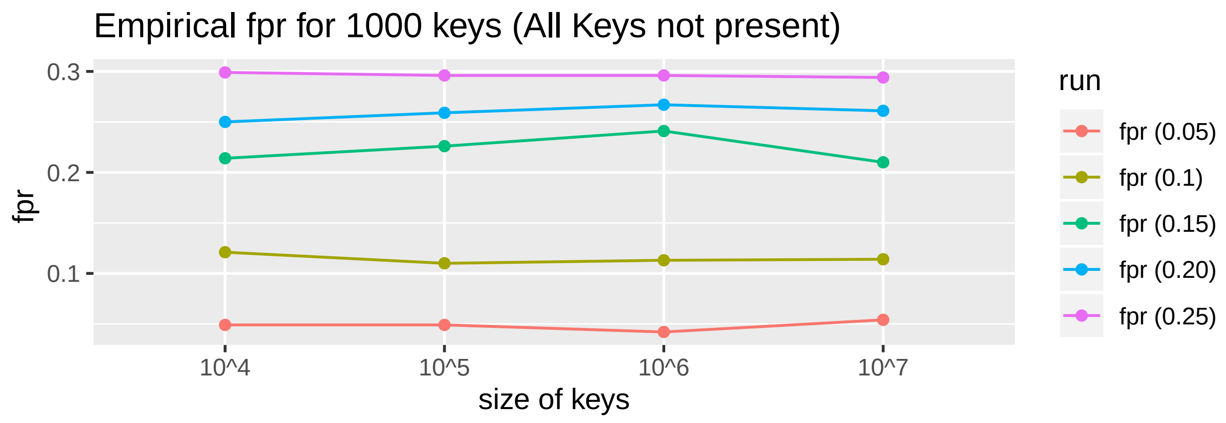
```r
plot3 <- ggplot(data=dmelt_pre, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("time (in milliseconds)") +
  labs(title="Average Query time across 1000 queries (All Keys present)")
# plot3

grid.arrange(plot1, plot2, plot3)
```

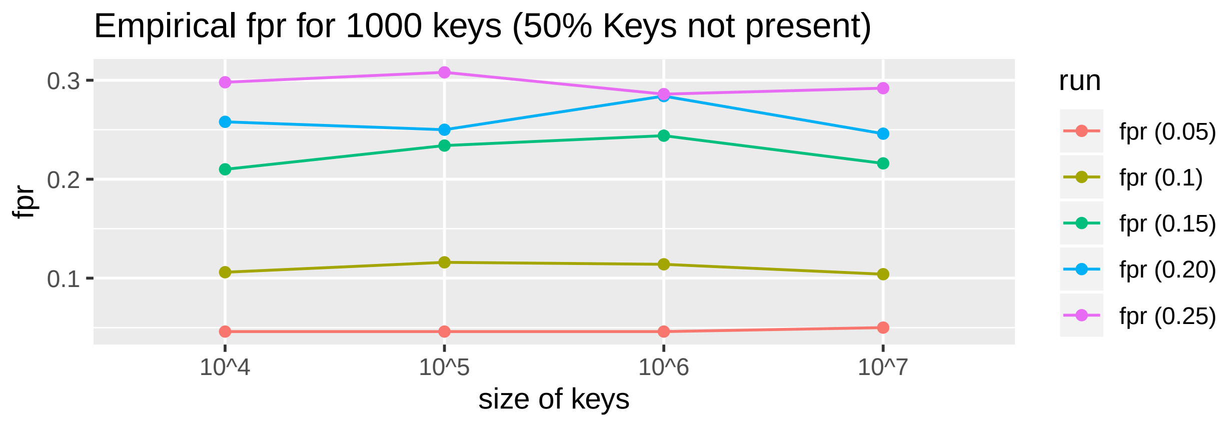


similar to the bloom filter, these plots show the average query time does not significantly increase as the size of the bloom filter increases. The plots shown here are the average query times for querying 1000 keys with the bloom filter. (top to bottom) 1) The first plot shows the average query time when no key in the search list is present in the bloom filter, 2) the second plot shows the average query time when 50% of the keys are present in the bloom filter and, 3) the third plot shows when all keys are present in the bloom filter. Irrespective of the search list, the average query time does not significantly change when the size of the bloom filter increases. it is independent of the size and the fpr.

False positive rate across 1000 queries

```
times_abs <- rjson::fromJSON(file="bbf/absfpr.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_abs) <- xlabels
times_abs <- as.data.frame(times_abs)
colnames(times_abs) <- xlabels
times_abs$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_abs <- melt(times_abs, id.vars = c("run"))
dmelt_abs$value <- dmelt_abs$value / 1000

plot1 <- ggplot(data=dmelt_abs, aes(x=variable, y=value, group=run, color=run)) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("fpr ") +
  labs(title="Empirical fpr for 1000 keys (All Keys not present)")
# plot1

times_mix <- rjson::fromJSON(file="bbf/mixfpr.json")
xlabels <- paste0("10^", seq(from=4, to=7))
names(times_mix) <- xlabels
times_mix <- as.data.frame(times_mix)
colnames(times_mix) <- xlabels
times_mix$run <- paste0("fpr", c(" (0.05)", " (0.1)", " (0.15)", " (0.20)", " (0.25)"))
dmelt_mix <- melt(times_mix, id.vars = c("run"))
dmelt_mix$value <- dmelt_mix$value / 500

plot2 <- ggplot(data=dmelt_mix, aes(x=variable, y=value, group=run, color=run)) +
  # geom_errorbar(aes(ymin=value-sd, ymax=value+sd), width=.1) +
  geom_line() +
  geom_point() +
  xlab("size of keys") +
  ylab("fpr") +
  labs(title="Empirical fpr for 1000 keys (50% Keys not present)")
# plot2

grid.arrange(plot1, plot2)
```





These plots show the empirical fpr for two different scenarios, 1) top, where no keys from the search list are present in the bloom filter and 2) bottom where only 50% of the keys are present in the bloom filter. Blocked bloom filters have a higher fpr than the traditional/normal bloom filter but its a significant effect. Except for the bloom filter with theoretical fpr 0.15, the rest of the experiments showed only a minor increase in the empirical fpr. I am trying to understand why this is the case (probably my selection of the keys or could be something else)