Problem 2

HW 4

Hanyuan Chi(chixx105), Zhi Shen(shenx704)

February 26, 2017

```
suppressPackageStartupMessages({
  library(knitr)
  library(memoise)
  library(assertthat)
  library(dplyr)
  library(ggplot2)
})
```

Dynamic Programming

You are working in an industry research lab and you are asked to assign scientists to teams.

There are 4 separate research teams right now working on the same project *independently* and each has its own probability of failure. Now your company tells you that they will hire 3 extra scientists and the company asks you to assign the scientists to teams such that the total probability of failure is minimized.

- Note the "total probability of failure" means the probability of *all* teams failing simultaneously.

The table of probabilities of failure for each team depending on the number of extra data scientists it gets is represented by this table

```
prob_fail_mat_ <- matrix(c(0.6, 0.8, 0.45, 0.75,
                           0.4, 0.5, 0.2, 0.45,
                           0.2, 0.3, 0.15, 0.3,
                           0.1, 0.2, 0.1, 0.15),
                         ncol = 4,
                         byrow = TRUE)

colnames(prob_fail_mat_) <- paste0("Team", 1:4)
rownames(prob_fail_mat_) <- paste0(0:3, " additional scientists")
kable(prob_fail_mat_)
```

	Team1	Team2	Team3	Team4
0 additional scientists	0.6	0.8	0.45	0.75
1 additional scientists	0.4	0.5	0.20	0.45
2 additional scientists	0.2	0.3	0.15	0.30
3 additional scientists	0.1	0.2	0.10	0.15

General Requirements

- Please review the resulting PDF and make sure that all code fits into the page. If you have lines of code that run outside of the page limits we will deduct points for incorrect formatting as it makes it unnecessarily hard to grade.

- Please make sure your PDF can be knitted in *at most* 3 minutes.

Question 1

Please solve the problem using Dynamic Programming recursive approach.

First plan then code. Think about optimality principle here. For this problem, you will need two indices for subproblems (just like 0-1 integer knapsack problem).

- Hint:
  - Try to use the logic similar (but not too similar!) to 0-1 knapsack:
    - with teams being analogous to items (j)
    - and extra scientists being analogous to bag capacity (s)
  - The underlying logic is not very different:
    - the more extra scientists you have the less chance that you fail (just like a bigger bag lets you leave fewer items behind)
    - having more teams lets you have a larger choice set to pick the most efficient project (just like having more available items would let a burglar fit the bag more fully and efficiently)
- Hint:
  - given that R has 1-based indexing instead of regular 0-based indexing used in other programming languages, I strongly advise you to use the function `prob_fail_mat_get()` that compensates for it. It is possible in our problem to have 0 extra people assigned to some team, while it is not possible to reference 0th element in a matrix in R.

```
prob_fail_mat_get <- function(ss,jj) (prob_fail_mat_[ss+1,jj])

prob_fail <- function(s,j) {
  if (!is.integer(j) || !is.integer(s))
    stop("s and j must be integers")

  if (s>=0 && j<=0L) return(1L)
  if (s<0L) return(Inf)

  prob_no_scientist_team_j <- prob_fail(s,j-1L)*prob_fail_mat_get(0L,j)
  prob_one_scientist_team_j <- prob_fail(s-1L,j-1L)*prob_fail_mat_get(1L,j)
  prob_two_scientist_team_j <- prob_fail(s-2L,j-1L)*prob_fail_mat_get(2L,j)
  prob_three_scientist_team_j <- prob_fail(s-3L,j-1L)*prob_fail_mat_get(3L,j)

  (min(prob_no_scientist_team_j,prob_one_scientist_team_j,
       prob_two_scientist_team_j,prob_three_scientist_team_j))
}

prob_fail(3L,4L)
```

```
## [1] 0.024
```

Question 2

Introduce memoization to make sure that your solution is really DP.

Please name your solution `mprob_fail`.

```
mprob_fail <- memoise( function(s,j) {
  if (!is.integer(j) || !is.integer(s))
    stop("s and j must be integers")
```

```
  if (s>=0 && j<=0L) return(1L)
  if (s<0L) return(Inf)

  prob_no_scientist_team_j <- mprob_fail(s,j-1L)*prob_fail_mat_get(0L,j)
  prob_one_scientist_team_j <- mprob_fail(s-1L,j-1L)*prob_fail_mat_get(1L,j)
  prob_two_scientist_team_j <- mprob_fail(s-2L,j-1L)*prob_fail_mat_get(2L,j)
  prob_three_scientist_team_j <- mprob_fail(s-3L,j-1L)*prob_fail_mat_get(3L,j)

  (min(prob_no_scientist_team_j,prob_one_scientist_team_j,
       prob_two_scientist_team_j,prob_three_scientist_team_j))
} )

mprob_fail(3L,4L)
```

```
## [1] 0.024
```

Unit tests for Q2

Let's take a look at the concept of a "unit test" in software engineering. The idea of the unit test is to confirm that a unit of our code (in this case, function `mprob_fail`) works as expected on particular inputs that we verified manually. I recommend using `assertthat` package in R. It offers a collection of functions like `validate_that()`, `see_if()` and `assert_that()` that are of great help when design unit tests as below.

For the purposes of this HW, I picked `validate_that()` instead of `assert_that()` since `assert_that()` will return an ERROR if assertion failed and thus, will not even let you produce the PDF until you fix the mistake. Instead, `validate_that()` will be nicer and will just report a message while still letting you knit the file.

Clearly, before you actually implement the code your unit tests will fail. But once you implement everything correctly they should all return `TRUE`.

```
# Clean the cache
forget(mprob_fail)

# When 0 extra scientists are available then fail probability is just the product
# of baselines for each team
validate_that(are_equal(mprob_fail(0L,4L),
                        prod(prob_fail_mat_[1,]) ))

# If 3 extra scientists are available and only team 1 is available
# then all 3 scientists must be go to team 1
validate_that(are_equal(mprob_fail(3L,1L),
                        prob_fail_mat_[3+1,1] ))
```

```
## [1] TRUE
## [1] TRUE
## [1] TRUE
```

Question 3

Solve the problem using bottom-up approach. Please fill out `prob_fail_` matrix that contains solutions to all the subproblems using bottom-up approach

- Hint:

- given that R has 1-based indexing instead of regular 0-based indexing used in other programming languages, I strongly advise you to use the function `prob_fail_get()` and `prob_fail_set()` that compensates for it. It is possible in our problem to have 0 extra people assigned to some team, while it is not possible to reference 0th element in a matrix in R.
- you may need a nested loop over both `s` and `j` variable as this problem requires traversing both indices.

```
prob_fail_ <- matrix(rep(NA_real_, 4*4), ncol=4)

# I recommend using these functions to set values of the matrix
# (to compensate for 1-based indexing)
prob_fail_get <- function(ss,jj) (prob_fail_[ss+1,jj])
prob_fail_set <- function(ss,jj, val) (prob_fail_[ss+1,jj] <<- val)

# Here goes your loop

for (s in 1:4)
{
  prob_fail_[s,1] <- prob_fail_mat_get(s-1,1)
}

for (j in 2:4) {
  for (s in 1:4){

    if (s == 1L)
      (prob_fail_[s,j] <- prob_fail_[s,j-1L]*prob_fail_mat_get(0L,j))
    if (s == 2L)
      (prob_fail_[s,j] <- min(prob_fail_[s,j-1L]*prob_fail_mat_get(0L,j),
                              prob_fail_[s-1L,j-1L]*prob_fail_mat_get(1L,j)))
    if (s == 3L)
      (prob_fail_[s,j] <- min(prob_fail_[s,j-1L]*prob_fail_mat_get(0L,j),
                              prob_fail_[s-1L,j-1L]*prob_fail_mat_get(1L,j),
                              prob_fail_[s-2L,j-1L]*prob_fail_mat_get(2L,j)))
    if (s == 4L)
      (prob_fail_[s,j] <- min(prob_fail_[s,j-1L]*prob_fail_mat_get(0L,j),
                              prob_fail_[s-1L,j-1L]*prob_fail_mat_get(1L,j),
                              prob_fail_[s-2L,j-1L]*prob_fail_mat_get(2L,j),
                              prob_fail_[s-3L,j-1L]*prob_fail_mat_get(3L,j)))

  }
}



# And finally show the result
prob_fail_get(3L,4L)
```

```
## [1] 0.024
```

Unit tests for Q3

As before, the idea of the unit test is to confirm that a unit of our code works as expected on particular inputs that we verified manually. I recommend using `assertthat` package in R. It offers a collection of functions like `validate_that()`,`see_if()` and `assert_that()` that are of great help when design unit tests as below.

Clearly, before you actually implement the code your unit tests will fail. But once you implement everything correctly they should all return `TRUE`.

```
# When 0 extra scientists are available then fail probability is just the product
# of baselines for each team
validate_that(are_equal(prob_fail_get(0L,4L),
                        prod(prob_fail_mat_[1,]) ))

# If 3 extra scientists are available and only team 1 is available
# then all 3 scientists must be go to team 1
validate_that(are_equal(prob_fail_get(3L,1L),
                        prob_fail_mat_[4,1] ))
```

```
## [1] TRUE
## [1] TRUE
```