BAN401 - Assignment

Problem 4

Problem 4 deals with decision making regarding efficient logistic financial schedule. So far the company uses diesel trucks to transport goods between several cities. The task of interest is to calculate the hypothetical savings by switching from diesel trucks to electric trucks for each route. The initial data for the routes is stored in a tibble.

```r
routes_df <-
  tibble::tibble(
    route = c("City A -> B", "City B -> C", "City A -> C"),
    distance = c(350, 500, 750),
    ferries = c(1, 2, 3),
    tolls = c(3, 5, 6)
  )
routes_df
```

```
# A tibble: 3 x 4
  route       distance ferries tolls
  <chr>          <dbl>   <dbl> <dbl>
1 City A -> B      350       1     3
2 City B -> C      500       2     5
3 City A -> C      750       3     6
```

A tibble is a modern data frame in the R-environment and offers plenty advantages according to create an efficient code structure. In this case we created a tibble called routes_df which contains data of the routes from City A to B, B to C and A to C and the given values of distance (in km), ferries (number of ferries) and tolls (number of toll stations).

The corresponding data of the different cost structure of diesel and electric trucks is stored in a list.

```r
vehicle_type_cost_list <-
  list(
    ET = c("Energy cost per km" = 0.2,
           "Price per ferry ride" = 180,
           "Toll cost per station" = 40),
    DT = c("Fuel cost per km" = 1.5,
           "Price per ferry ride" = 300,
           "Toll cost per station" = 80)
  )
vehicle_type_cost_list
```

```
$ET
   Energy cost per km  Price per ferry ride Toll cost per station
                  0.2                 180.0                  40.0

$DT
     Fuel cost per km  Price per ferry ride Toll cost per station
                  1.5                 300.0                  80.0
```

The list called vehicle_type_cost_list contains two elements ET (electric truck) and DT (diesel truck). These two elements do contain the costs of energy/fuel, price per ferry ride and toll cost per station.

The main challenge of this task is to apply mathematical operations while working with a data frame and a list at the same time. Therefore, we have to multiply the distance with the energy/fuel cost per km, the number of ferries with the price per ferry and the number of toll stations with the toll cost per station for each truck type and sum these multiplications up. Practically, this means that we have to multiply the first row of routes_df with the first element of vehicle_type_cost_list and so on. This creates an element-by-element multiplication. The last step then is to compute the sum of these multiplications.

The idea of our problem solution is to run a for-loop which iterates over the rows in routes_df and multiplies these elements with the given values in vehicle_type_cost_list.

Hence, we have to initialize empty vectors with three elements of data type "double" in order to store the results of the for loop.

```r
et_total <- vector(mode = "double", length = 3); et_total
```

```
[1] 0 0 0
```

```
dt_total <- vector(mode = "double", length = 3); dt_total
```

```
[1] 0 0 0
```

Now we can define the for-loop which fills the empty vectors with the representative values.

```
for(i in 1:nrow(routes_df)) {
  et_total[i] <- sum(routes_df[i,c(2,3,4)] * vehicle_type_cost_list$ET)
  dt_total[i] <- sum(routes_df[i,c(2,3,4)] * vehicle_type_cost_list$DT)
}
```

As declared above, we iterate over all columns in routes_df to extract the given information of all available routes. In this case we declare the iteration limit to three which is the number of rows in routes_df. Moreover, we access to the elements of interest of routes_df with square brackets. More precisely we want to gain access to the i-th row and all columns starting at column two. The first column is a character that contains the route direction and is therefore excluded according to the mathematical operations. These values will be then multiplied in an element by element manner with the elements stored in both objects ET and DT of vehicle_type_cost_list. The $-operator enables us to have direct access to the elements of these objects. Therefore we do not have to deal with an encrypted list. This multiplication creates a vector of three elements. Finally, the sum of these three elements will be calculated to receive the total cost of the specific route for each truck type.

```
et_total
```

```
[1] 370 660 930
```

```
dt_total
```

```
[1] 1065 1750 2505
```

As we suggested, the results are now stored in the initially defined vectors et_total and dt_total element by element.

These vectors are now available in the global environment. Hence, we append them to the initial data frame routes_df by storing et_total and dt_total as new columns.

```
routes_df$et_total <- et_total
```

```r
routes_df$dt_total <- dt_total
```

As we can see, transporting the goods by diesel trucks is consistently more expensive than by electric trucks.

```r
routes_df
```

```
# A tibble: 3 x 6
  route       distance ferries tolls et_total dt_total
  <chr>          <dbl>   <dbl> <dbl>    <dbl>    <dbl>
1 City A -> B      350       1     3      370     1065
2 City B -> C      500       2     5      660     1750
3 City A -> C      750       3     6      930     2505
```

These findings can now be highlighted by calculating the savings by switching from diesel trucks to electric trucks. Hence, we create a new column in routes_df named savings by the total costs of transportation with diesel trucks minus the transportation costs by electric trucks.

```r
routes_df$savings <- routes_df$dt_total - routes_df$et_total
routes_df
```

```
# A tibble: 3 x 7
  route       distance ferries tolls et_total dt_total savings
  <chr>          <dbl>   <dbl> <dbl>    <dbl>    <dbl>   <dbl>
1 City A -> B      350       1     3      370     1065     695
2 City B -> C      500       2     5      660     1750    1090
3 City A -> C      750       3     6      930     2505    1575
```

Problem 5

Problem 5 deals with an allocation problem. In particular, a retailer faces the issue that he/she needs to procure goods worth exactly $200 to replenish their stock for the next business cycle. Therefore, he has a set of given items:

```
items_df <-
  tibble::tibble(
  items = c("small item", "medium item", "large item", "extra-large item",
           "luxury item", "premium item", "bulk pack"),
  costs = c(1,2,5,10,20,50,100)
  )
items_df
```

```
# A tibble: 7 x 2
  items            costs
  <chr>            <dbl>
1 small item           1
2 medium item          2
3 large item           5
4 extra-large item    10
5 luxury item         20
6 premium item        50
7 bulk pack          100
```

We defined the set of given items as items_df which calls the tibble()-function of the R-package tibble in order to create a modern data frame. items_df possesses two columns. The first column defines the given items and the second contains the corresponding costs in USD.

The main task regarding establishing the R-code is to create a function that returns all possible purchase combinations of the given items. That means we have to explore multiple combination paths under the restriction of reusable items. A very efficient approach to solve this problem is initializing a recursive function that calls itself. This approach enables us to create a dynamic program to find all possible purchase combinations.

```
purchase_combinations <- function(target, costs, index) {

  if (target == 0)
    return(1)

  if (target < 0 | index < 1)
    return(0)

  include_current_item <- purchase_combinations(target - costs[index],
                                                costs,
                                                index)
  exclude_current_item <- purchase_combinations(target,
                                                costs,
                                                    index - 1)

  combinations <- as.numeric(
    include_current_item + exclude_current_item)

  return(combinations)
}
```

The function is defined with three arguments. The first argument initializes the target spend amount. The second argument includes the cost structure of the given items and the third argument provides access to the indices of the items. We have to include the index of items in order to realize all possible purchase combinations. A given item possibly can be reused to accomplish the target spend amount. Hence, the index is highly important.

The problem solution deviation is defined with the general structure of recursive functions. Therefore, we want to go through the function chunk wise.

The first chunk determines a binary system where we work with 0 and 1 according to the enumeration of possible purchase combinations. If the target is equal to zero then 1 will be returned. If the target is less than zero or there are no more items left then 0 will be returned. This code chunk satisfies the conditional manner of the recursive function.

To provide the explanation of the second code chunk, we will start with an initial example. Let us assume that the target is equal to 200, the costs and the indeces are given through the data frame items_df. Since the target initially is greater than zero and the starting item does have the index seven, there will be the 0 returned because we did not find a representative purchase combination since the purchase volume is not used up. This means that we can call the purchase_combinations()-function itself in order to deduct the given item costs. Here, the costs of the seventh item will deduct the target spending volume. This process will iterate until the target spending volume is reached. Hence, a purchase combination path is found.

This expresses that if there is a positive residual target value, then the costs of the given item will be recognized and deduct the target volume. If the value is overstretched then the item will be excluded in the purchase path. All possible combinations of item allocation will be summed up and returned in the last instance.

```
total_combination_number <-
  purchase_combinations(target = 200,
                        costs = items_df$costs,
                        index = length(items_df$costs))

cat("How many item combinations add up to $200?", "\n", "-> There are",
    total_combination_number, "unique item purchase combinations.")
```

```
How many item combinations add up to $200?
 -> There are 73681 unique item purchase combinations.
```

In order to show the number of all possible combinations, we have to provide initial values to the arguments. Here, we declared the target as $200, the costs as the predefined items costs of the data frame items_df and the index argument due to the length of items_df. As we can see, there are 73681 possible purchase combinations that reach the target spending value of $200.