COL215 ASSIGNMENT 3 REPORT

In this assignment, we extended the code of previous assignment so as to delete the terms whose corresponding K-map regions are fully contained within one or more other terms/regions. So, we used the logic that if a particular cell containing 1 is covered by a single region then that corresponding region need not be deleted.

The list named list_k is the output list of the previous assignment function. We created a list “ONE” in which we appended a lists containing terms contained in the particular region of list_k. Therefore, ONE is a list of lists. For creating “ONE”, we traversed through list_k and func_TRUE, if the element of list_k is contained in the element of func_TRUE then that element is appended in list “one” and finally one is appended in ONE.

Next, we created a list dump to store the region to be deleted (which should not go in the output list). The output list will contain the terms to be given as output. Now we traversed through the list ONE to check if all the elements of a particular sublist are contained in other sublists of ONE and if this is true then the corresponding region goes in dump list, otherwise in output list. In this loop, we have used one helper function named list_check which returns the common elements in two lists.

The rest commented code is Demo code which prints Deleted regions, deleted cells, covering region of the deleted terms.

The time complexity of our algorithm is O(nm^3 + m^3)

Below are the testcases that we have run:

The testcases are made for 2, 3, 4, 5, 6, 7, 8 input K-map and we have covered all the base as well as edge test cases.

We have also covered the case when a particular region is covered by more than one other regions and also the case when that region is covered by a single other region.

```
print(opt_function_reduce(["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'","a'bc'd'e", "a'bc'de", "a'bc'
print(opt_function_reduce(["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"],["abc'd"]))
print(opt_function_reduce(["a'b'c'd", "a'b'cd", "a'bc'd", "abc'd'", "abc'd", "ab'c'd'", "ab'cd"],["a'bc'd'
print(opt_function_reduce(["a'b'c'd'e'", "a'bc'd'e'", "abc'd'e'", "ab'c'd'e'", "abc'de'", "abcde'","a'bcde
print(opt_function_reduce(["a'b'", "a'b", "ab'", "ab"], []))
print(opt_function_reduce["a'b'c", "a'bc", "a'bc'", "ab'c'"],["a'c", "a'b", "ac'"])
print(opt_function_reduce(["a'b'c'd'e'", "a'bc'd'e'", "abc'd'e'", "ab'c'd'e'", "abc'de'", "abcde'",
"a'bcde'", "a'bcd'e'", "abcd'e'", "a'bc'de", "abc'de", "abcde",
"a'bcde", "a'bcd'e", "abcd'e", "a'b'cd'e", "ab'cd'e"],[]))
print(opt_function_reduce(["a'b'c", "a'bc", "a'bc'", "ab'c'"], ["abc'"]))
print(opt_function_reduce(["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"], ["abc'd"]))
print(opt_function_reduce(["a'b'c'd", "a'b'cd", "a'bc'd", "abc'd'", "abc'd", "ab'c'd'", "ab'cd"],["a'bc'd'
print(opt_function_reduce(["a'b'c'd'e'", "a'bc'd'e'", "abc'd'e'", "ab'c'd'e'", "abc'de'", "abcde'", "a'bcd
print(opt_function_reduce(['abcde', "ab'c'de", "ab'cde"], ["ab'c'de'"]))
print(opt_function_reduce(["a'b'c'", "a'b'c", "a'bc'", "a'bc"], ["ab'c'", "abc'", "ab'c", "abc"]))
print(opt_function_reduce(["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'", "a'bc'd'e", "a'bc'de", "a'bc
print(opt_function_reduce(["a'bc'defgh", "a'bc'd'efgh", "abc'd'efgh", "abc'defgh", "a'bc'def'gh", "a'bc'd'
print(opt_function_reduce(["a'b", "a'b'"], []))
print(opt_function_reduce( ["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'", "a'bc'd'e", "a'bc'de", "a'b
# ----------------------------------------TEST CASES FOR 2 VARIABLES----------------------------------
print(opt_function_reduce(["a'b'"], []))
print(opt_function_reduce(["a'b'"], ["a'b"]))
print(opt_function_reduce(["a'b'", "ab'"], ["a'b"]))
# ALL ARE ONE
print(opt_function_reduce(["a'b'", "a'b", "ab'", "ab"], []))
# ----------------------------------------TEST CASES FOR 3 VARIABLES----------------------------------
print(opt_function_reduce(["a'b'c'", "a'b'c", "a'bc", "a'bc'"], ["ab'c'", "ab'c"]))
```

```
print(opt_function_reduce(["a'b'c", "a'bc", "a'bc'", "ab'c'"], ["abc'"]))
print(opt_function_reduce(["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"], ["abc'd"]))
print(opt_function_reduce(["a'b'c'd", "a'b'cd", "a'bc'd", "abc'd'", "abc'd", "ab'c'd'", "ab'cd"],["a'bc'd'
print(opt_function_reduce(["a'b'c'd'e'", "a'bc'd'e'", "abc'd'e'", "ab'c'd'e'", "abc'de'", "abcde'", "a'bcd
print(opt_function_reduce(['abcde', "ab'c'de", "ab'cde"], ["ab'c'de'"]))
print(opt_function_reduce(["a'b'c'", "a'b'c", "a'bc'", "a'bc"], ["ab'c'", "abc'", "ab'c", "abc"]))
print(opt_function_reduce(["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'", "a'bc'd'e", "a'bc'de", "a'bc
print(opt_function_reduce(["a'bc'defgh", "a'bc'd'efgh", "abc'd'efgh", "abc'defgh", "a'bc'def'gh", "a'bc'd'
print(opt_function_reduce(["a'b", "a'b'"], []))
print(opt_function_reduce( ["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'", "a'bc'd'e", "a'bc'de", "a'b
# ----------------------------------------TEST CASES FOR 2 VARIABLES----------------------------------
print(opt_function_reduce(["a'b'"], []))
print(opt_function_reduce(["a'b'"], ["a'b"]))
print(opt_function_reduce(["a'b'", "ab'"], ["a'b"]))
# ALL ARE ONE
print(opt_function_reduce(["a'b'", "a'b", "ab'", "ab"], []))
# ----------------------------------------TEST CASES FOR 3 VARIABLES----------------------------------
print(opt_function_reduce(["a'b'c'", "a'b'c", "a'bc", "a'bc'"], ["ab'c'", "ab'c"]))
print(opt_function_reduce(["a'b'c'", "a'b'c"], ["ab'c'", "ab'c"]))
print(opt_function_reduce(["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'", "a'bc'd'e", "a'bc'de", "a'bc
print(opt_function_reduce(["a'b'c'd'e'f'", "a'b'c'd'e'f", "a'b'cd'e'f'", "a'b'cd'e'f", "a'bc'd'e'f'", "a'b
print(opt_function_reduce(["a'b'c'd'e'f'", "a'b'c'd'e'f", "a'b'cd'e'f'", "a'b'cd'e'f", "a'b'cd'ef'", "a'b'
print(opt_function_reduce(['abcd', "ab'c'd", "ab'cd"], ["abcd'"]))
print(opt_function_reduce(["a'bc'de", "a'bcde"], ["a'b'cde", "a'b'cde'"]))
print(opt_function_reduce(["a'bcdef", "a'b'c'def", "a'b'c'd'e'f'", "a'b'c'd'ef", "a'b'c'd'e'f", "a'b'cdef"
print(opt_function_reduce(['abcde', "ab'c'de", "ab'cde"], ["ab'c'de'"]))
```