COL215 Assignment 3

- Report by Piya Bundela(2021CS10118) and Aastha Rajani(2021CS10093)

Assignment 3 of col215 makes us use knowledge of K-maps. Here we are provided two lists, one which contains the cells which have 1 in the k map and other which contains the cells which have ‘x’ in the k map. We are asked to output the maximum expanded regions corresponding to each cell containing 1.

Here, input is in form of two list, fun_true- that has cells containing 1 and fun_dc – that contains cells with x. We are supposed to return regions that cover all the cells containing 1

In previous part of assignment, we were supposed to return maximum region corresponding to each fun_true element which was in list called Final. We here have a list called answer_list that has elements of Final but, with complemented literals in a string to their corresponding uppercase literals.

We now check the elements of list fun_true that are contained in each region and make a list of it for each region of answer_list and further append it list called major_list.

We will now take intersection of first element of major_list with all other elements and then take union of all intersections.

If this union is equivalent to the element, it would imply that cells covered by this region is covered in other regions as well, so we do not require this region, we send it to discard_list.

If this union is not equivalent to the element, it would imply that this region has cells that are not covered by other regions.

We will repeat this for all element of major_list except that we will now not take intersection with elements that we have added to discard_list.

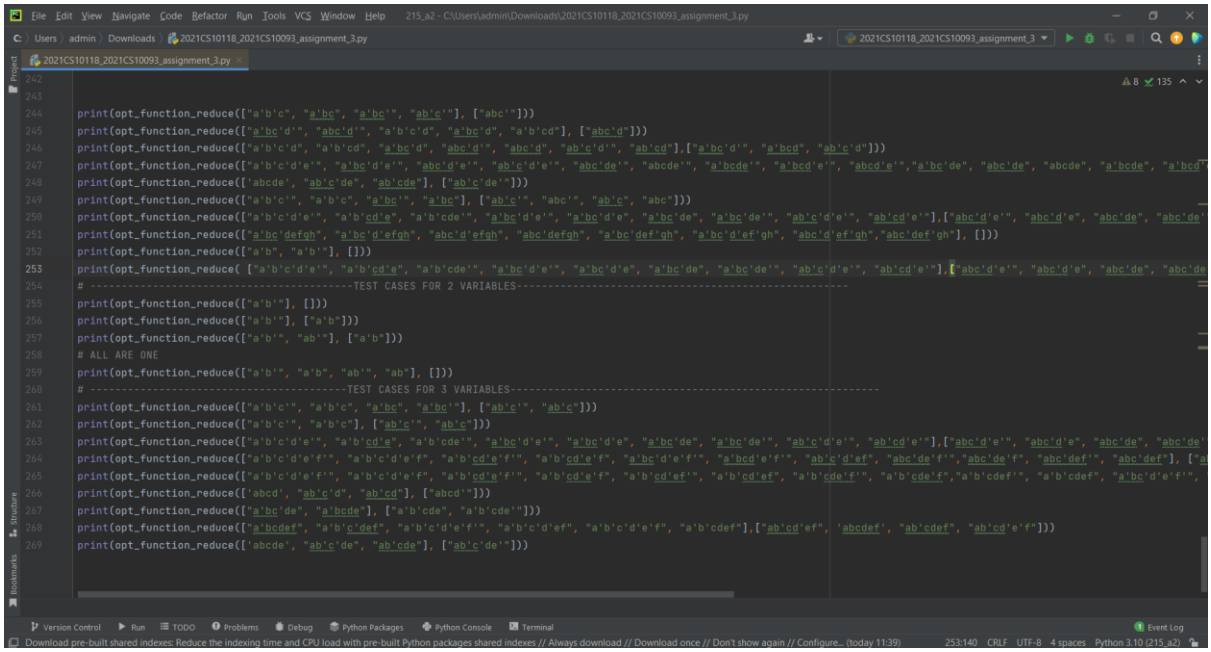
we put elements of answer_list corresponding to major_list that are not in discard_list to list called FINAL. This list has regions that contains all 1s, ensuring that each cell is contained in at most 1 region. In the end we return the converted form of FINAL.

Time Complexity = $mn^3 + n^3$

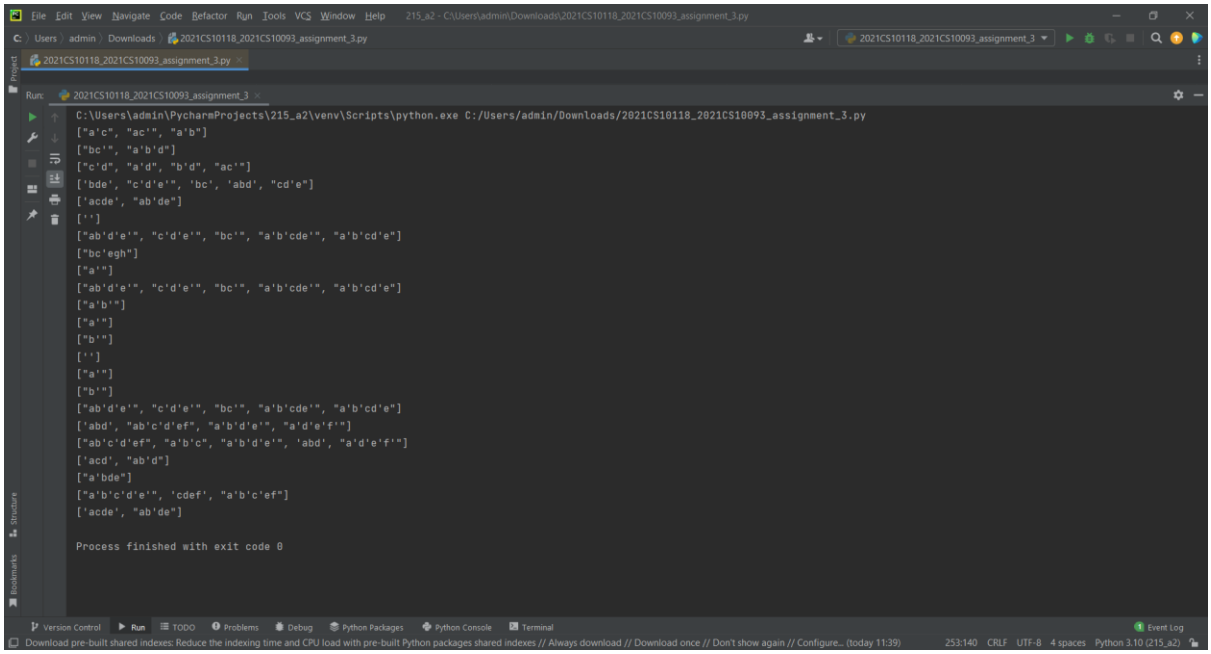
Where m is number of variables and n is number of elements in fun_True + number of elements in fun_Dc.

what test cases did you run? Why is this set good enough to validate the implementation?

We have checked several cases like when some overlapping regions need to be deleted, the cases which cover entire of the k map and cases in which every region is non-redundant...our code outputs the correct regions for such cases.



```
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'", "a'bcd", "ab'c'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'", "a'bcd'e'", "abcd'e'","a'bc'de", "abc'de", "abcde", "a'bcde", "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'de'", "ab'c'd'e'", "ab'cd'e'"],["abc'd'e'", "abc'd'e", "abc'de", "abc'de'
print(opt_function_reduce(["a'bc'defgh", "a'bc'd'efgh", "a'bc'd'efgh", "abc'd'efgh", "abc'defgh", "a'bc'def'gh", "a'bc'd'ef'gh", "abc'd'ef'gh","abc'def'gh"], []))
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'bc'de'", "ab'c'd'e'", "ab'cd'e'"],["abc'd'e'", "abc'd'e", "abc'de", "abc'de
# ----------------------------------------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'de'", "ab'c'd'e'", "ab'cd'e'"],["abc'd'e'", "abc'd'e", "abc'de", "abc'de
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'bcd'e'f'", "ab'c'd'ef", "abc'de'f'","abc'de'f", "abc'def'", "abc'def"], ["a
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'cd'ef", "a'b'cde'f'", "a'b'cde'f","a'b'cdef'", "a'bc'd'e'f'",
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"],["ab'cd'ef", 'abcdef', "ab'cdef", "ab'cd'e'f"]))
print(opt_function_reduce(['abcde', "ab'c'de", "ab'cde"], ["ab'c'de'"]))
```



```
C:\Users\admin\PycharmProjects\215_a2\venv\Scripts\python.exe C:/Users/admin/Downloads/2021CS10118_2021CS10093_assignment_3.py
["a'c", "ac'", "a'b"]
["bc'", "a'b'd"]
["c'd", "a'd", "b'd", "ac'"]
['bde', "c'd'e'", 'bc', 'abd', "cd'e"]
['acde', "ab'de"]
['']
["ab'd'e'", "c'd'e'", "bc'", "a'b'cde'", "a'b'cd'e"]
["bc'egh"]
["a'"]
["ab'd'e'", "c'd'e'", "bc'", "a'b'cde'", "a'b'cd'e"]
["a'b'"]
["a'"]
["b'"]
['']
["a'"]
["b'"]
["ab'd'e'", "c'd'e'", "bc'", "a'b'cde'", "a'b'cd'e"]
['abd', "ab'c'd'ef", "a'b'd'e'", "a'd'e'f'"]
["ab'c'd'ef", "a'b'c", "a'b'd'e'", 'abd', "a'd'e'f'"]
['acd', "ab'd"]
["a'bde"]
["a'b'c'd'e'", 'cdef', "a'b'c'ef"]
['acde', "ab'de"]

Process finished with exit code 0
```