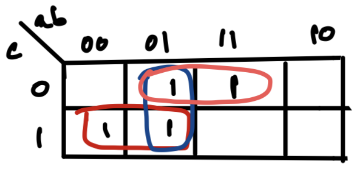
COL215P: Software Assignment - 3
Deleting Unnecessary Terms

Chinmay Mittal (2020CS10336) and Dhruv Tyagi(2020CS10341)

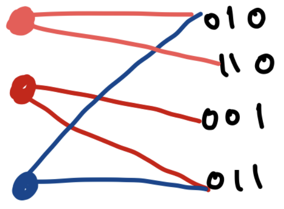
Explanation of Algorithm

We build on the previous assignment, we had a bfs traversal over all expansions to find the optimal expansions for each term. In the previous assignment if there were multiple prime implicants for a term we returned any one of them. In this assignment the graph traversal returns all the prime implicants for a term i.e. all the nodes at the lowest possible depth for each term. They represent the optimal (i.e. largest) regions enclosing a term.
This way we create the prime implicant chart where for each term we have the corresponding prime implicants i.e. maximal and optimal regions enclosing them. Out of all these prime implicants across all terms we have to choose the minimum number of prime implicants to cover all the terms.
We first process the essential prime implicants these corresponds to those terms which have only one optimal region enclosing them it is thus essential to take them.
Once we have processed the prime implicant chart by removing all the essential prime implicants and the terms which they cover we get a chart where there are prime implicants remaining (not essential) and terms being covered by them. We have to choose a minimum subset of these prime implicants such that all terms are covered. This corresponds to the Set-Cover problem which is NP-Hard and no optimal polynomial time algorithm is know for the same.
One algorithm is Petrick's Method which gives an optimal solution to this problem however it takes time which is exponential in the size of the input and it doesn't work in time for cases when there are large number of input variables. There are several polynomial time approximate algorithms available for the set cover based on heuristics.
The one we implement is maximum remaining nodes covered. We first the pick the prime implicant which covers the most number of terms then we pick the prime implicant which covers most number of remaining uncovered terms and so on. We keep doing this till all the terms are covered. Although not optimal this algorithm has a polynomial running time and also has a O(logn) approximation bound to the optimal algorithm.
Thus there are some prime implicants which we don't consider because the terms they cover are covered by a combination of other prime implicants. We print this for the demo.

We have also submitted patrick's method which is optimal but exponential in running time.



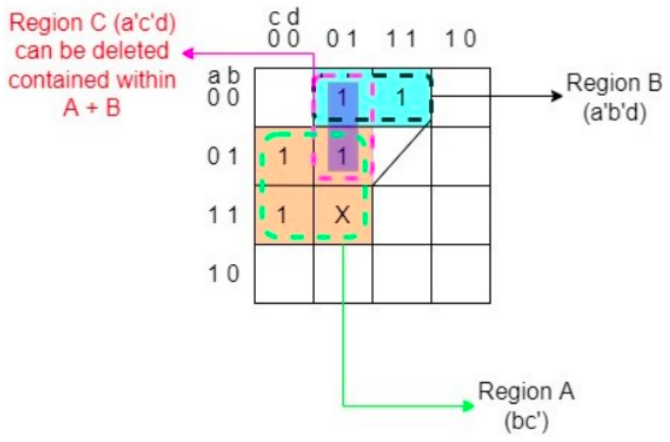
SET COVER FORMULATION.



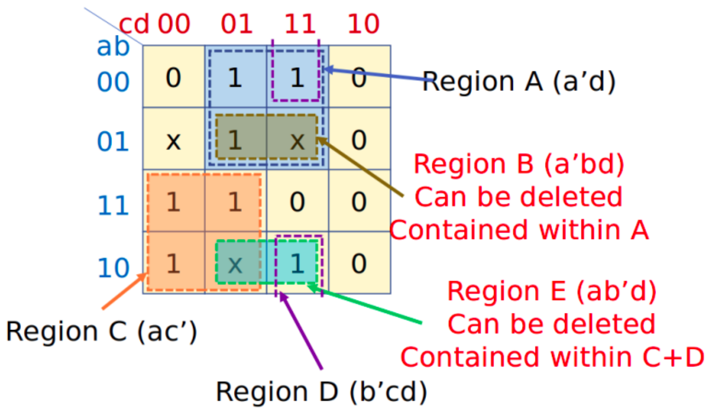
Time Complexity

As show in the previous assignment the worst case running time to get all valid regions from the bfs is O($M^{1.58}$ logM) where M is the size of the input list. Once we know all legal regions we can do a dfs from each node to get maximal and optimal regions which enclose it (prime implicants), this takes O(M*$M^{1.58}$ logM) (we do a dfs for each source).
Once we have all the prime implicants we make the prime implicant chart and perform the heuristic for the set cover problem.
Let us say that we have X terms and Y prime implicants.
Then in the first iteration we compute how many terms are covered by each prime implicant. It takes O(X) time to compute the number of terms covered by a prime implicant and we do this for each prime implicant thus the first iteration takes O(YX) time.
Similarly in the next iteration we have to find amongst Y-1 remaining implicants which covers the largest number of terms this will take O((Y-1)X) iterations and so on for Y-2, Y-3 ….. till 0 when we have no remaining implicants. Summing these up we get that the total number of iterations for this set cover heuristic is O(Y^2X). Here X is O(M) where M is the size of the input list (number of terms) and Y is also O(M) (the maximum number of prime implicants is bounded by the number of terms).
This shows that our algorithm is polynomial in size of the input list with worst case running time complexity being O(M^3) here M is the size of the input list. Hence our algorithm is polynomial in the size of the input and can work for decently large sized inputs giving a solution which though not optimal is bounded by a log(N) approximation factor.

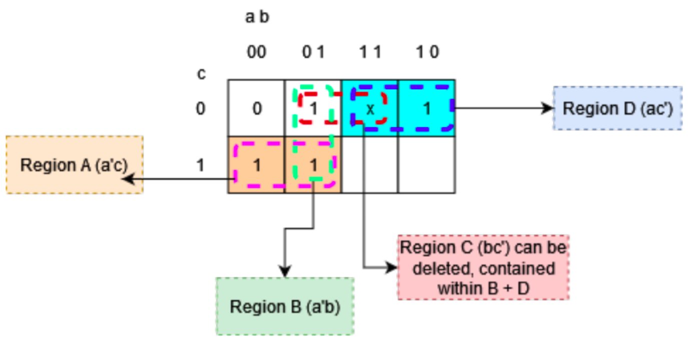
Test Cases



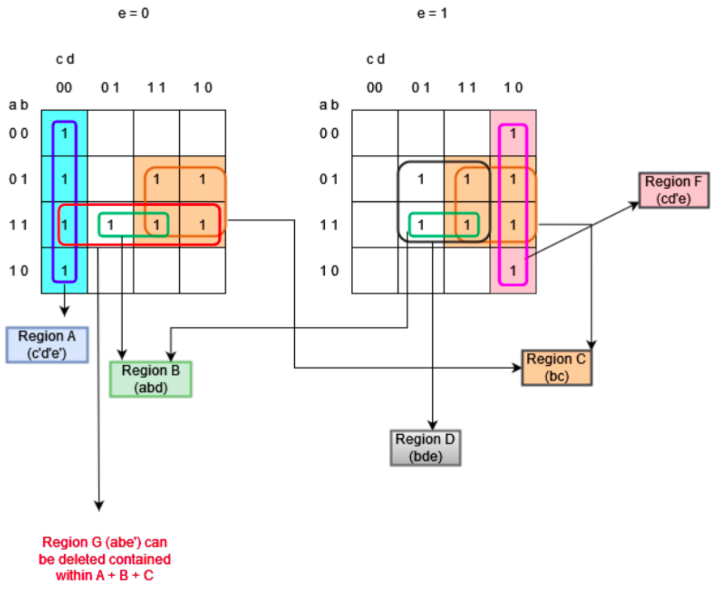
```
func_TRUE = ["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"]
func_DC = ["abc'd"]
Output => ["bc'", "a'b'd"]
```



```
func_TRUE = ["a'b'c'd", "a'b'cd", "a'bc'd", "abc'd'", "abc'd", "ab'c'd'",
"ab'cd"]
func_DC = ["a'bc'd'", "a'bcd", "ab'c'd"]
Output => ["a'd", "ac'", "b'd"] (here we take region b'd instead of b'cd)
```



```
func_TRUE = ["a'b'c", "a'bc", "a'bc'", "ab'c'"]
func_DC = ["abc'"]
Output => ["bc'", "a'c", "ac'"] (here we delete region a’b instead of bc’)
```



```
func_TRUE = ["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"]
func_DC = []
Output => ["abe'", "c'd'e'", 'bc', 'bde', "cd'e"]
```

```
This test covers the case when only essential prime implicants come in the
answer
```

```
func_TRUE = ["abc'", "ab'c'", "a'b'c", "a'bc"]
func_DC = []
Output => ["ac'", "a'c"]
```

```
This covers the case when there are no essential prime implicants
func_TRUE = ["a'bc'", "abc'", "a'bc", "abc"]
func_DC = []
Output => [b]
```

```
Using do not care terms for essential prime implicant
func_TRUE = ["a'bc'd'", "ab'c'd'"]
func_DC = ["a'b'c'd'", "abc'd'", "a'bc'd", "a'bcd", "a'bcd'", "ab'c'd",
"ab'cd", "ab'cd'"]
Output => [“c’d’”]
```

```
func_TRUE = ["abc'd", "a'bcd", "abcd'"]
func_DC = ["abc'd'", "a'bc'd", "abcd"]
Output => ['bd', 'ab']
```

```
Prime Implicants being repeated in minterms (6 variable case)
func_TRUE = ["ab'cd'e'f'", "ab'c'd'e'f'", "abcd'e'f", "ab'cd'e'f",
"abc'd'ef", "abcd'ef", "a'bc'd'ef'", "abc'd'ef'", "a'bcdef'", "a'bc'def'",
"a'b'cdef", "a'bcdef", "a'b'c'de'f", "a'b'cde'f", "a'b'c'de'f'"]
func_DC = []
Output => ["acd'e'f", "abc'd'e", "a'bc'ef'", "a'bcde", "a'b'cdf",
"abcd'f", "ab'd'e'f'", "a'b'c'de'"]
```

```
This covers the case when the largest region is not in the answer
func_TRUE = ["a'bc'd", "abc'd"]
func_DC = ["abc'd'", "ab'c'd'", "ab'c'd", "abcd", "ab'cd", "abcd'",
"ab'cd'"]
Output = ["bc'd"]
```

```
8 variable case
func_TRUE = ["a'b'c'de'f'g'h", "a'b'cd'e'f'g'h", "a'bc'd'e'fgh'",
"a'bc'd'e'fgh", "a'bc'd'e'fg'h", "a'bc'd'e'fg'h'", "a'bc'd'efg'h'",
"a'bc'de'fg'h", "a'bcde'fg'h", "abc'd'e'fg'h", "abc'de'fg'h"]
```

```
func_DC = ["a'b'c'd'e'f'g'h'", "a'b'c'd'e'f'g'h", "a'b'c'd'e'f'gh",
"a'b'c'd'e'f'gh'", "a'b'c'de'f'g'h'", "a'b'c'de'f'gh", "a'b'c'de'f'gh'",
"a'b'cd'e'f'g'h'", "a'b'cd'e'f'gh", "a'b'cd'e'f'gh'", "a'bcd'e'f'g'h'",
"a'b'cde'f'g'h", "a'bcd'e'f'gh", "a'b'cde'f'gh'"]
Output => ["a'b'c'e'f'", "a'b'd'e'f'", "a'bc'd'e'f", "a'bc'd'fg'h'",
"a'bde'fg'h", "bc'e'fg'h"]
```

```
This case shows that our algorithm can give suboptimal answers
func_TRUE = ["a'b'c'de'f'g'h", "a'b'cd'e'f'g'h", "a'bc'd'e'fgh'",
"a'bc'd'e'fgh", "a'bc'd'e'fg'h", "a'bc'd'e'fg'h'", "a'bc'd'efg'h'",
"a'bc'de'fg'h", "a'bcde'fg'h", "abc'd'e'fg'h", "abc'de'fg'h",
"abc'd'efgh", "abc'd'efgh'", "abc'd'ef'gh'", "abc'd'ef'gh",
"abc'd'ef'g'h", "abc'def'g'h", "abcdef'g'h", "abcd'ef'g'h",
"ab'cd'ef'g'h", "ab'cd'ef'gh", "ab'cd'ef'gh'", "ab'cd'efgh'",
"ab'cd'efgh", "abcd'efgh", 'abcdefgh', "abc'defgh", "abcdefgh'",
"abcdef'gh'", "abcdef'g'h'", "abc'def'gh'", "abcd'ef'gh'"]
func_DC = ["a'b'c'd'e'f'g'h'", "a'b'c'd'e'f'g'h", "a'b'c'd'e'f'gh",
"a'b'c'd'e'f'gh'", "a'b'c'de'f'g'h'", "a'b'c'de'f'gh", "a'b'c'de'f'gh'",
"a'b'cd'e'f'g'h'", "a'b'cd'e'f'gh", "a'b'cd'e'f'gh'", "a'bcd'e'f'g'h'",
"a'b'cde'f'g'h", "a'bcd'e'f'gh", "a'b'cde'f'gh'"]
Output => ["ab'cd'ef'h", "abcdegh'", "abcdef'g'", "a'b'c'e'f'",
"a'b'd'e'f'", "a'bc'd'e'f", "a'bc'd'fg'h'", "a'bde'fg'h", "bc'e'fg'h",
"abc'd'eg", "abef'g'h", "ab'cd'eg", 'abefgh', "abef'gh'"]
```