Assignment 2

Binary Decision Diagram

In this assignment I have implemented Reduced ordered Binary Decision Diagram – BDD. My BDD will take as an input a string containing a Boolean function and create a tree using Shannon decomposition of the given Boolean function. After creation it will return the root node of the tree. I choose to implement this assignment in python programming language

Tree body

Text

Description automatically generated

This is my constructor of the BDD tree. It contains root of the tree. Order of Boolean arguments that determine how the tree will be created using Shannon decomposition. A dictionary – used\_nodes – where all unique nodes are stored. Number of unique nodes and number of unique Boolean function arguments.

Node body

Text

Description automatically generated

Every node that is created contains its parent, pointer to left\_child that will contain lower node and pointer to right\_child that will contain higher node based on Shannon decomposition. Every node also contains number of unique variables in current function. Variable count is later used in function .use() where we traverse the tree.

Tree function: create(„Boolean function“, [order of arguments])

How to use the create function:



Firstly you need to create a variable that will be an object BDD(). Then you can call the function use and insert arguments. Arguments for boolean function needs to only contain uppercase Alphabet letters, ! indicating negation of the boolean variable. If you want to multiple variables just place them next to each other without any kind of character.

Example: „AB+C+!DE+!F!G“

After you create the correct boolean function, the second argument is list of the order of inserted boolean variables for Shannon decomposition.

Example: [‚A‘,‘B‘,‘C‘,‘D‘,‘E‘,‘F‘,‘G‘]

Reduction

Merge isomorphic nodes

In BDD tree we encouter isomorphic nodes commonly. Those nodes take up additional memory so we want to eliminate them. I eliminate them by checking if current created node has been already created. If yes then i dont create current node but point to that already created node in memory. Code:

Text

Description automatically generated

In the body of my tree class I store a hash map(used\_nodes). Hash map stores all unique nodes created. By checking if given node is in hash map I know if i need to create a node or just point to already created node. This function is always called when new node is beeing created. Node creation example:



Eliminate redundant nodes

When nodes left and right child are the same, the current node is redundant. The way I eliminate redundant nodes is that I set the nodes parent pointer to node left or right child. This basically skips the redundant node and saves memory

Text

Description automatically generated

This function checks if nodes children are equal. If yes it means given node is redundant and returns True.

Text

Description automatically generated

In my create function I call check\_redundant and based on its result I eliminate redundant node. Once a node is reduced i call private create function on current node which is now None to create its children.

Time complexity

The best case scenario is O(n) where n is the length of boolean function. This occurs when boolean function is either a tatology or a contradiction. The worst case scenarion is O(2m x n) where m is the number of unique variables and n is the lenght of boolean function. This occurs when no reduction takes place and the boolean function is traversed for each node.

Testing

For testing correctness of my BDD I implemented my own evaluate function which is located in main.py and demo\_main.py. For my time and reduction test I genereted 100 random boolean functions. Boolean functions are made from 5,9,13,14 and 15 unique variables. Example: „AD+BC+C+DC+EA“ contains 5 unique variables. Genereted functions always contain atleast one set of unique variables. For each of 100 i tested every possible use combination rangind from 25 possibilities to 215 possibilities. Each time using my own eveluate function to check if created tree return correct use value. All tests passed.

Test results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num of unique variables \*1 | one function (seconds) \*2 | creation time (seconds) \*3 | use time (seconds) \*4 | Average reduction (%) \*5 |
| 5 | 0.0219066143 | 0.0000587714 | 0.0000015629 | 81.42857142 |
| 9 | 0.0449365925 | 0.0000856374 | 0.0000046893 | 99.21989552 |
| 13 | 10.1895639371 | 0.0012277522 | 0.0000030200 | 99.59280961 |
| 14 | 24.7490310144 | 0.0014945092 | 0.0000029472 | 99.66210706 |
| 15 | 68.3901624218 | 0.0020699575 | 0.0000032781 | 99.85933408 |

Test table legend

1 – :column represents from how many unique variables generated boolean function consists of

2 – Test1:column represents average time for creating, using and evaluating one boolean function with all possible use combinations.

3 – Test2: column represents average time for creation of tree with number of unique variables in left column.

4 – Test3:column represents the average time for using(traversing the tree) to get the result of a boolean function with given use values.

5 – Test4: column represents average reduction of the tree. Reduction = (number of nodes with reduction) / (number of nodes without reduction). I get nodes with reduction by getting the length of dictionary with unique nodes from my tree. And number of nodes without reduction is calculated by (2n+1 -1) where n represents number of unique variables in boolean given boolean function

Comparison

Test1 – all combinations

(seconds)

(unique var)

This test results make sense. Since checking the tree for 5 variables is nothing compare to checking the tree for 15 variables which is 215 possible combinations. For this test I am once again remainding that these data take into account use,create and evaluate function.

Test 2 – creation time

(seconds)

This test result again show promising values. As we need more unique variables the tree needs to create more nodes. Then do all the reduction function to make it as small as possible. My creation function is called recursively so more and more recursion have to happen in order to create the tree

Test 3 – use time

(seconds)

(unique var)

This test result again show promising values except values from testing 9 unique variables. Worst time complexity for the use function is O(n). We have to iterate n times to get return value from the tree. So values should be pretty similar. I have not figured out why the usage of trees with 9 unique variables is slowest. The only explonation I have is that I have genereted my own random boolean function and test values for 9 variables might have been more complicated than other randomly generated functions

Test 4 - reduction

(seconds)

(unique var)

These results are what we theoretically expect. When more unique variables are given more nodes are created. Even more already creatde nodes are created. So by not actually creating them but just setting pointers to them we reduce the number of nodes. For example the last row of not reduced BDD contains 2n nodes with values 0 and 1. By reducing them we save a lot of memory.

Evaluate function

Text

Description automatically generated

This the function evaluate that I used in every test to check if my created tree returned correct values.

Conclusion

Almost all test that were conducted showed results that were theoretically correct. The reduction works as it should and average times are also looking promising except test 3 with strange values from tree 9 unique variables.