## Description of Algorithm

The input for buildcompactbdt is a list of string, where each entry in string corresponds to a row in truth table which evaluate to 1. This input can be considered as an POS (Product of Sum) expression, where each string corresponds to a minterm, with the left most side as x1, which corresponds to LSB (Least significant bit) of the minterm, as illustrated by the diagram below.

Input string

| Most left | … | Most right |
|---|---|---|
| $x_1$ | … | $x_n$ |

$\Leftrightarrow$

Internal representation (term)

| MSB | … | LSB |
|---|---|---|
| $x_{n-1}$ | … | $x_0$ |

To simplify the binary decision tree, instead of approach where seems most forward, to generate a tree then rotate and remove repetitive node.
I decide to per-process the string in term of simplify SOP expression, remove "don't cares" and get (essential) prime implicants, which is the simplest form possible, if done as binary decision graph, there will be using least node possible.
But the requirement of this assignment is binary decision tree, so I need to turn the graph into tree by creating duplicate node, with the least amount of duplication possible.

## Buildcompactbdt

This function takes one argument, which is input strings (fvalues) and the return with root node of the decision tree. Below are the descriptions of the algorithm

1. Convert vector of string to vector of minterm

2. Using Quine–McCluskey[1,2] algorithm to find prime implicants.
   By using this method, up to this step, the output is optimal, there are no possible simplification to further reduce the term.

3. (optional)[3] Using chart(heuristic) or Petrick's method[4] to find essential prime implicants
   By using Petrick's method will ensure the number of term is optimal, but a much faster approached is to use heuristic method to approximate the essential prime implicants. However, as shown in later experimental section, this step does not produce a much different for the node of binary decision tree.

4. Using heuristic algorithm to generate the binary decision tree, with aim to make duplicated node appear as deeper node as possible.
   By using 3 conditions in order of propriety to choice the node, firstly choice node with most don't care, secondly choice node with most unbalanced 1 and 0, finally choice by numerical order from node 1 to node n.

---

[1]Donald Krambeck (2016) *Everything About the Quine-McCluskey Method* https://www.allaboutcircuits.com/technical-articles/everything-about-the-quine-mccluskey-method/ [ 29th/04/2018]

[2] corporate author (2018) *Quine–McCluskey algorithm* https://en.wikipedia.org/wiki/Quine-McCluskey_algorithm [ 29th/04/2018]

[3] This step was considered at the start, but I had chosen not to implement due to the level of optimisation and time constraint. Please refer to 1st part of Experimental Evaluation for the experiment.

[4]Donald Krambeck (2016) *Prime Implicant Simplification Using Petrick's Method* https://www.allaboutcircuits.com/technical-articles/prime-implicant-simplification-using-petricks-method/ [ 29th/04/2018]

Below is an example of work through, for input fvalues of 0000(0), 1000(1), 0100(2), 1010(5), 0110(6), 1110(7), 0001(8), 1001(9), 0101(10) and 0111(14).

The website "allaboutcircuits"[5] showed how to simplify the fvalues above with Quine–McCluskey algorithm to obtain essential prime implicants so I don't repeat it here. At start of step 4, the table below shows the representation of prime implicants.

| essential prime implicants | Internal representation in struct implicant | | |
|---|---|---|---|
| $x_1x_2x_3x_4$[6] | $x_4x_3x_2x_1$ | mask | Minterm |
| 1-10 | 01-[7]1 | 0010 | 0101 |
| -00- | -00- | 1001 | 0000 |
| 01-- | --10 | 1100 | 0010 |

In step4, 3 condition are used for this heuristics algorithm, listed below in the order of how decision is taken:

i. choice the $x_n$ with least don't care, to make duplicated node appear as deeper node as possible,
ii. If there are multiple $x_n$ with same number of don't care, additional screening is used to choose the node with most unbalanced 1 and 0. Such that for example with internal representation ($x_4x_3x_2x_1$) of 0001 and 0010, node $x_3$ then $x_4$ will be chosen as it is most unbalanced. Reference the full example from "Testing on case where can't be merged" in Evaluation section.
iii. If there are multiple $x_n$ have same ranking from condition i and ii, then numerical order is choice.

Continue with the example, reference the table above, $x_3, x_{2,} x_1$ have the least don't care ("-" represented as 1 in mask), and same unbalance of 1 and 0 (all have 1 ones and 2 zeros). Which means numerical order will be used, which choice $x_1$ as root of the tree.

When $x_1$ is chosen as root node, the table above can be split into 2.

| When $x_1$ is 0, left child | | | When $x_1$ is 1, right child | | |
|---|---|---|---|---|---|
| $x_4x_3x_2x_1$ | mask | Minterm | $x_4x_3x_2x_1$ | mask | Minterm |
| -00(0)[8] | 1000 | 0000 | 01-(1) | 0010 | 0101 |
| --1(0) | 1100 | 0010 | -00(1) | 1000 | 0000 |

[5]Donald Krambeck (2016) *Everything About the Quine-McCluskey Method* https://www.allaboutcircuits.com/technical-articles/everything-about-the-quine-mccluskey-method/ [ 29th/04/2018]

[6]$x_1x_2x_3x_4$ is the order of the input string fvaules, but internally is stored as $x_4x_3x_2x_1$. Later on will always using internal representation unless when evaluate against fvalue(s).

[7] "-" here means don't care, 01-1 is same as $\overline{x_4}x_3x_1$

[8] Bracket means this bit ($x_1$ in this case) had been envaulted, the mask and minterm at this bit will be normalised to 0

Further consider node (bit) not been evaluated, using same heuristic strategy, $x_2$ will be chosen, as $x_4, x_3, x_2$have don't care 2,1,0 respectively, for the internal node for left child of root node $x_1$.By the same method, $x_3$ will be chosen as internal node for the right node. Which produces a table as shown below.

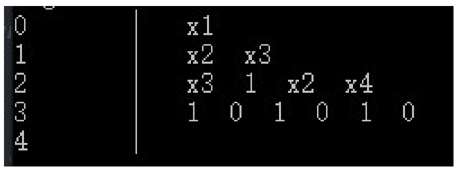
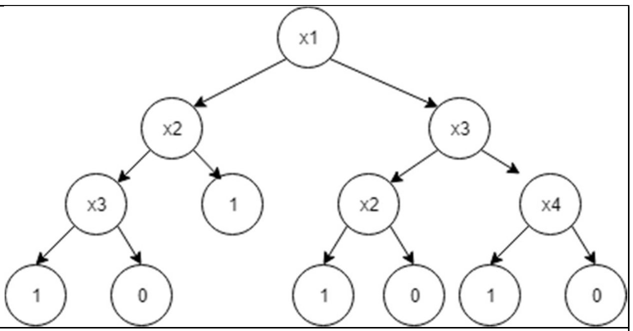
| When $x_1$ is 0, left child | | When $x_1$ is 1, right child | |
|---|---|---|---|
| When $x_2$ is 0 | When $x_2$ is 1 | When $x_3$ is 0 | When $x_3$ is 1 |
| $x_4x_3x_2x_1$ | $x_4x_3x_2x_1$ | $x_4x_3x_2x_1$ | $x_4x_3x_2x_1$ |
| -0(0)(0) | --(1)(0) | -(0)0(1) | 0(1)-(1) |

By repeat steps above, it will reach 2 kind of base case. First base case is when there is no implicant in the table, second base case is when there is *one* implicant with non-evaluated bit are all don't care ("-"). For example, when $x_1$is 1,$x_3$ is 0.

| When $x_1$ is 1 and $x_2$ is 0 | | | | | |
|---|---|---|---|---|---|
| When $x_2$ is 0 | | | When $x_2$ is 1 | | |
| $x_4x_3x_2x_1$ | mask | Minterm | $x_4x_3x_2x_1$ | mask | Minterm |
| -(0)(0)(1) | 1000 | 0000 | | | |

As shown on table above, on left side when $x_2$ is 0, the only term $x_4$ is not evaluated which is don't care. This node has label 1 and no child node, represented in Boolean is $\overline{x_3}\ \overline{x_2}\ x_1$ evaluate to 1.
On the other hand, on left side when $x_2$ is 1, there are no term as the table content. This node has label 0 and no child node, represented in Boolean is $x_3\ \overline{x_2}\ x_1$ evaluate to 0.

The binary decision tree produced by this method is shown below

|  |  x1<br>x2<br>x3<br>x3<br>1<br>x2<br>x4<br>1<br>0<br>1<br>0<br>1<br>0 |
|---|---|
| *output from my print tree function | *graphical enhanced by hand |

```
0        x1
1        x2  x3
2        x3  1  x2  x4
3        1  0  1  0  1  0
4
```

By using this method, only 13 nodes is required, compare to a full binary decision tree as did in assignment 1, requires 31 nodes.