ROBDD Implementation

Goal: * Build an ROBDD for any given Boolean expression (e.g. $E = x.y + \bar{z}$)

Strategy: * Each Boolean operator is implemented as a function (e.g. BDD AND(BDD, BDD);).

* A single shared BDD is built incrementally by successively applying the Boolean-operator functions (e.g. $E = OR(\underbrace{AND(x,y)}_{①}, \underbrace{NOT(z)}_{②})$)

* A single table is used to keep track of all the nodes generated so far ③ ⇒ no redundant nodes are created ⇒ no isomorphic trees exist ⇒ each step results in an RBDD

Node Table (NTable)

* A single table is used to keep track of all the variables and their order ⇒ each step results in an OBDD

Variable Table (VTable)

Trick: * All Boolean operator can be expressed using two functions:

Ⓐ BDD Cofactor (BDD F, Variable x, Bool k)

Cofactor(F, x, 1) ≡ F_x , Cofactor(F, x, 0) ≡ $F_{\bar{x}}$

Ⓑ BDD ITE (BDD I, BDD T, BDD E)

(If Then Else)

ITE(I, T, E) ≡ $I.T + \bar{I}.E$

* Given these two functions (i.e., Cofactor and ITE), The Boolean operator functions can be implemented as follows:

  - BDD AND (BDD A, BDD B) { return ITE(A,B,0); }
  - BDD OR (BDD A, BDD B) { return ITE(A, 1, B); }
  - BDD NOT (BDD A) { return ITE(A, 0, 1); }
  - BDD ⇒ (BDD A, BDD B) { return ITE(A, B, 1); }
  - BDD ⇔ (BDD A, BDD B) { return ITE(A, B, NOT(B)); }
  - BDD FORALL (Variable x, BDD A) {
    return AND(Cofactor(A, x, 1), Cofactor(A, x, 0)); }
  - BDD EXISTS (variable x, BDD A) {
    return OR(Cofactor(A, x, 1), Cofactor(A, x, 0)); }

Datastructures:

* Variable ≡ string (or int or ....!)

* Node ≡ < Variable x, Node * hi, Node * lo >
  - ZERO := < "ZERO", NULL, NULL >
  - ONE := < "ONE", NULL, NULL >

* NTable ≡ Node[] ← list of nodes
  - Initially containg only two node ZERO & ONE.
  - Every entry (element) represents a unique node
  - methods:
    - Node * (Get (Node) n) ≡ ① lookup NTable for node n
      ② If found, return * to entry
      ③ Else, add entry for n, return * to entry.

* VTable ≡ Variable [] ← "ordered" list of variables

⊛ Methods:

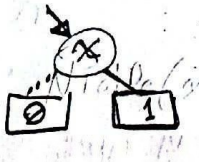
- int Get (Variable x) ≡
  ① lookup VTable for variable x.
  ② If found, return order (index) of x.
  ③ Else, add entry for x and return order (index) of x.
- Variable TopMost (BDD []) ≡
  return the top most variable from among all the variables that show up in a list of BDDs.

* BDD ≡ Node *

BDD Constructor(s):

⊛ BDD Build (Bool k) { if (k==0) return NTable.Get(ZERO);
else return NTable.Get(ONE); }

⊛ BDD Build (Variable x) {

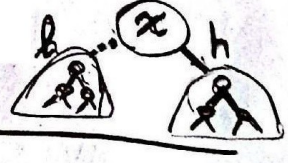


VTable.Get(x);
return NTable.Get(< x, Build(1), Build(0) >); }

⊛ BDD Build (Variable x, BDD h, BDD l) {

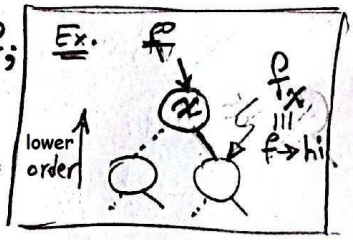
raise exception

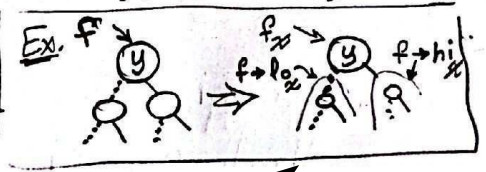
if (VTable.Get(x) ≥ VTable.Get(VTable.TopMost([h,l]))) return NULL;
if (h==l) return h;
return NTable.Get(< x, h, l >); }



Cofactor:

```
BDD Cofactor ( BDD f , Variable x , Bool k)
{
  if ( f == NTable.Get(ZERO) || f == NTable.Get(ONE) ) return f;
  if(VTable.Get(x) < VTable.Get( VTable.TopMost( [f] ))) return f;
  if(VTable.Get(x) == VTable.Get(VTable.TopMost([f])) && k == 1)
        return f->hi;
  if(VTable.Get(x) == VTable.Get(VTable.TopMost([f])) && k == 0)
        return f->lo;
  return Build (VTable.TopMost([f]),
          Cofactor(f->hi, x, k),
          Cofactor(f->lo, x, k) );
}
```





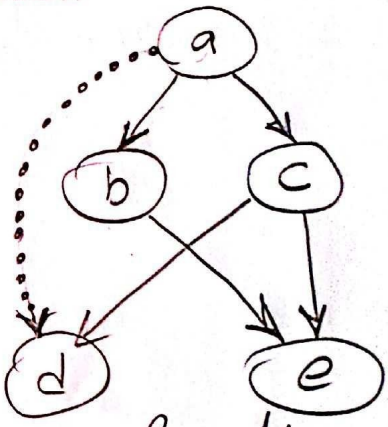
ITE:

```
BDD ITE (BDD i, BDD t, BDD e)
{
  /* Terminal Case(s) */
  if (i == NTable.Get(ONE)) return t;
  if (i == NTable.Get(ZERO)) return e;
  if ( t == e ) return t;
  if ( ..... ) return ...;
  /* General Case */
  x = VTable.TopMost([i, t, e]);
  f_x = ITE(Cofactor(i, x, 1), Cofactor(t, x, 1), Cofactor(e, x, 1));
  f_x' = ITE(Cofactor(i, x, 0), Cofactor(t, x, 0), Cofactor(e, x, 0));
  return Build(<x, f_x, f_x'>)
}
```

Drawing BDD's

* We can use a s/w package named "GraphViz"
  - "Graphviz" supports different types of graphs: directed, undirected, layouts, ...
  - "dot" is the "Graphviz" command used for drawing directed-acyclic graphs (such as BDD's).
  - "dot" takes as input a file containing a graph represented in a simple textual language called "DOT", and generates a drawing in a graphical format such as "PS".
  - "DOT" example:

```
digraph G {
    a -> b -> e;
    a -> c;
    c -> {e; d}
    a -> d [style = dotted];
}
```



* To draw BDD's, we can create a function that converts the BDD datastructure into "DOT" file.

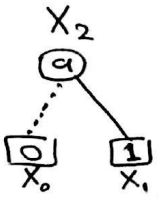
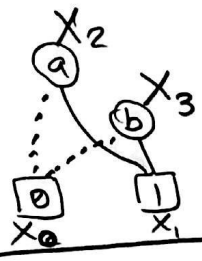
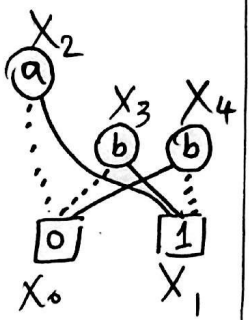
```
void BDD2DOT(BDD x, file_handle f)
{ if (x == NTable.Get(ONE))
    - Add a square node labeled "1" to file f
  else if (x == NTable.Get(ZERO))
    - Add a square node labeled "0" to file f
  else
    ① Add a circular node labeled as the root node of x [not needed!]
    ② Add a solid edge between root of x and root of x->hi
    ③ Add a dotted edge between root of x and root of x->lo
    ④ BDD2DOT(x->hi, f)
    ⑤ BDD2DOT(x->lo, f)
}
```

* NOTE: Don't forget to add to file f: "digraph G {" and "}"!

Example: Show all the steps required to construct an ROBDD to represent $a + \bar{b}$

Solution:

- The ROBDD is constructed by the following function call: OR(Build(a), NOT(Build(b)))
  where a = "a" & b = "b"
- This results in the following function call:
  ITE(Build(a)⑦, Build(1)①, ITE(Build(b)②, Build(0)⑥, Build(1)③)④⑤)

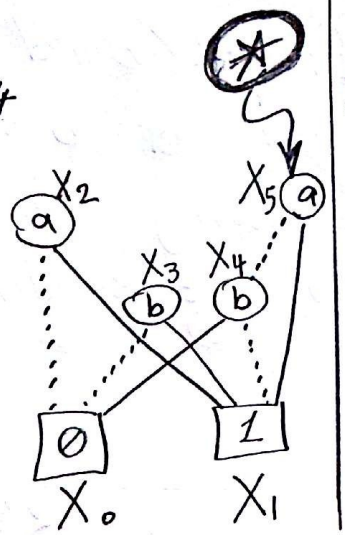
STEP	NT	VT
① Build(a) NT.Get(a, Build(1), Build(0)) $= X_2$ 	<"ZERO", NULL, NULL> @X_0 <"ONE", NULL, NULL> @X_1 <"a", X_1, X_0> @X_2	"a" → 1
② Build(1) $= X_1$		
③ Build(b) NT.Get(b, Build(1), Build(0)) $= X_3$ 	<"b", X_1, X_0> @X_3	"b" → 2
④ Build(0) $= X_0$		
⑤ Build(1) $= X_1$		
⑥ ITE(Build(b)$_{X_3}$, Build(0)$_{X_0}$, Build(1)$_{X_1}$) Topmost x = "b" $f_x = ITE(X_3 \xrightarrow{1} hi, X_0, X_1) = X_0$ $f_{\bar{x}} = ITE(X_3 \xrightarrow{0} lo, X_0, X_1) = X_1$ 	<"b", X_0, X_1> @X_4	

⑦ ITE(Build(a), Build(1), ITE(.....))
X_2 X_1 X_4

Top most

$$\begin{cases} x = \text{"a"} \\ f_x = ITE(\overbrace{X_2 \to hi}^{1}, \overbrace{X_1}^{1}, X_4) = X_1 \\ f_{\bar{x}} = ITE(\overbrace{X_2 \to lo}^{0}, \overbrace{X_1}^{1}, X_4) = X_4 \end{cases}$$

$= X_5$



$\langle \text{"a"}, X_1, X_4 \rangle$
@X_5