Project #3 Report

Technology Mapping

Problem

An exercise of technology mapping

Part VI. (10 pts) You are to map the circuit shown in Figure 4(a). Find the mapping of the circuit with the minimum cost using the cell library of Figure 4(b). **Show your work!**

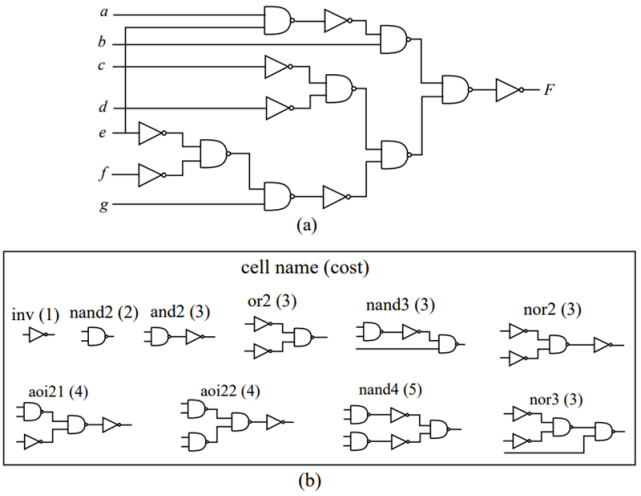
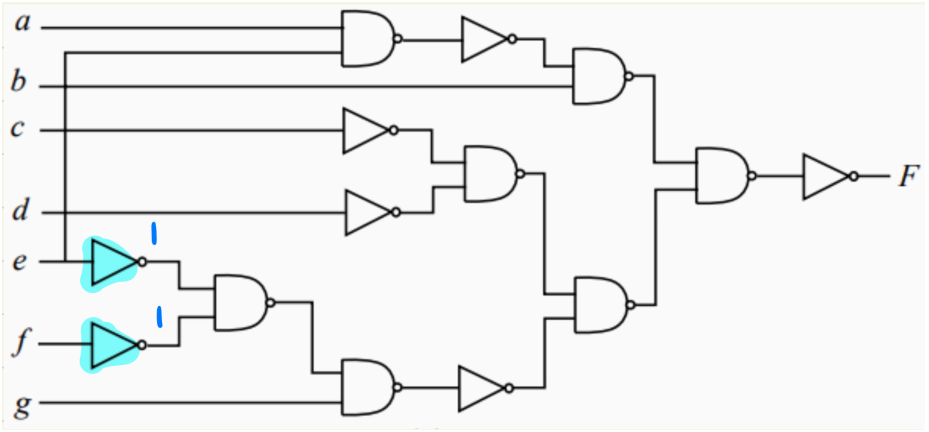


Figure 4: A cell library and a circuit.

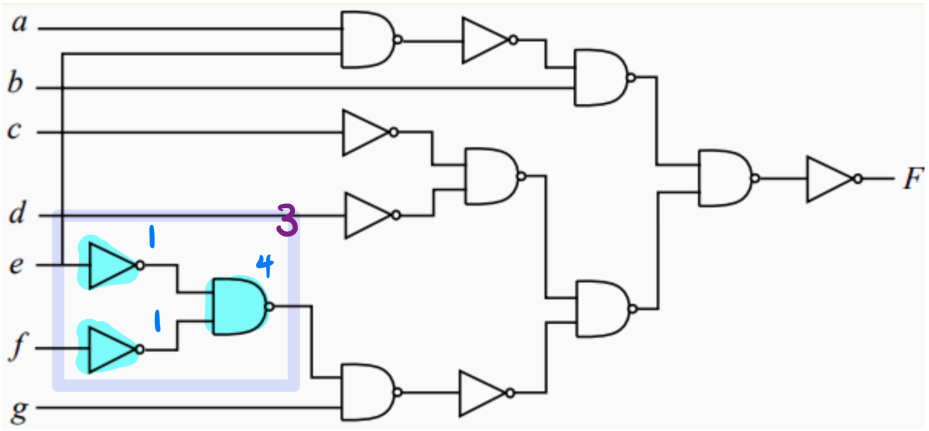
Solution

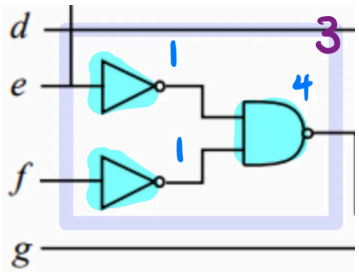
1. **Optimal tree covering 1**



e → inv(1) ; f → inv(1) ⇒ Optimal subtree covering cost = 1

2. **Optimal tree covering 2**

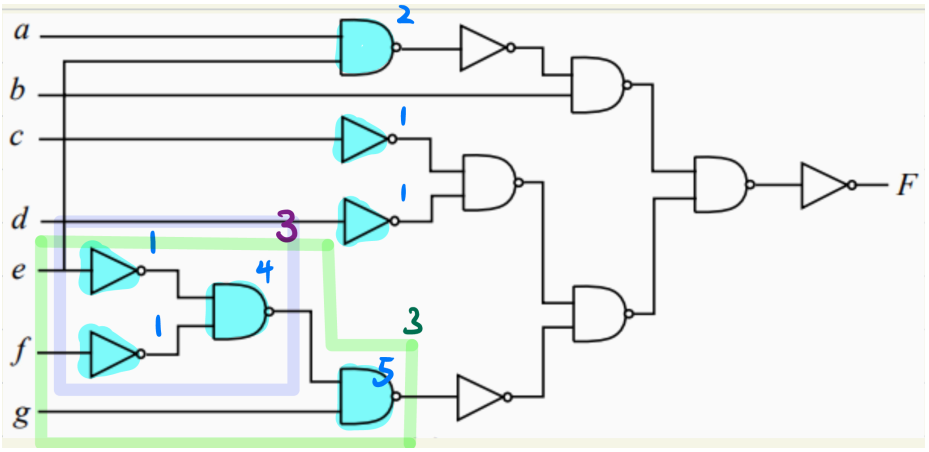




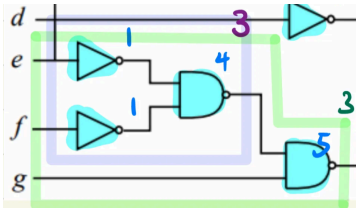
Part 1

- Option A: nand2 (2)+subtree (1) + subtree (1) = 4
- **Option B: or2 (3) = 3**

3. **Optimal tree covering 3**



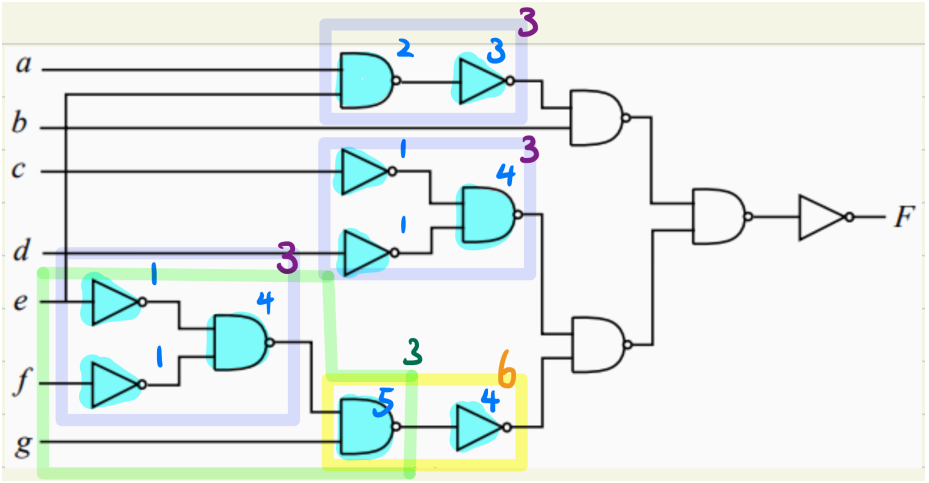
a & e → nand2 (2) ; c → inv(1) ; d → inv(1)

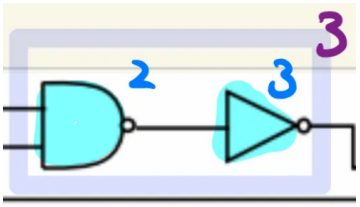


Part 1

- Option A: nand2 (2)+subtree (3) = 5
- **Option B: nor3 (3) = 3**

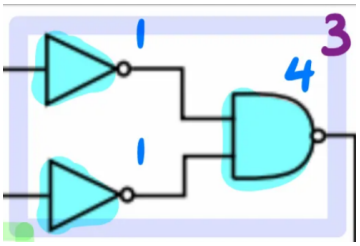
4. **Optimal tree covering 4**





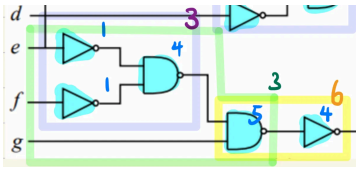
Part 1

- **Option A: inv (1)+subtree (2) = 3**
- **Option B: and2 (3) = 3**



Part 2

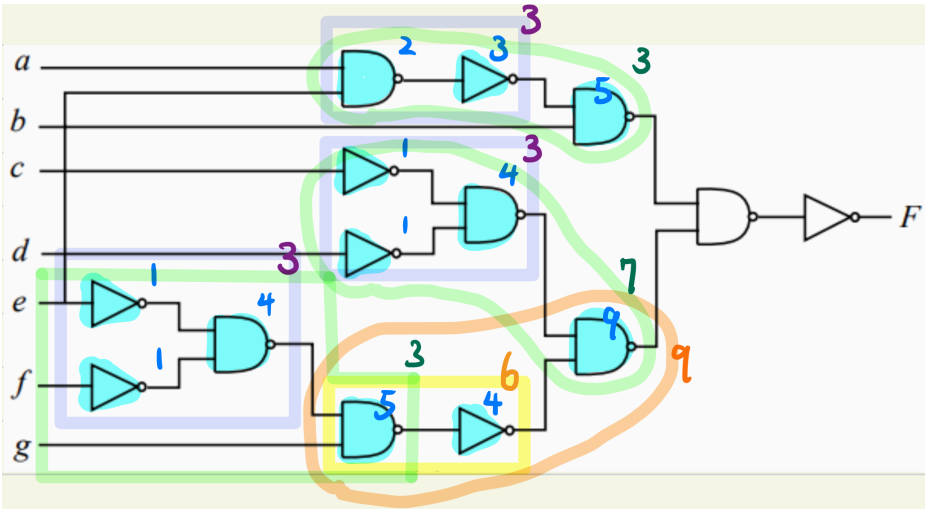
- Option A: nand2 (2)+subtree (1) + subtree (1) = 4
- **Option B: or2 (3) = 3**

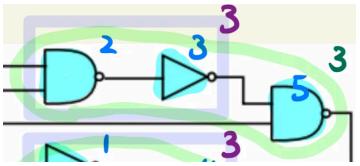


Part 3

- **Option A: inv (1)+subtree (3) = 4**
- Option B: and2 (3) + subtree (3) = 6

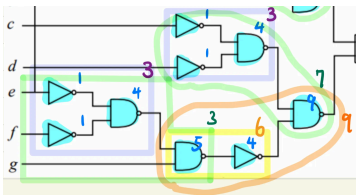
5. **Optimal tree covering 5**





Part 1

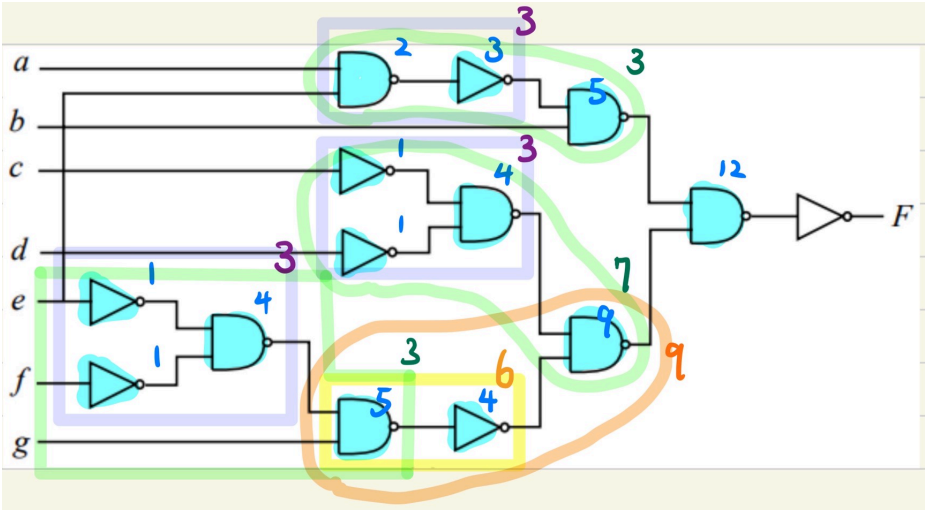
- Option A: nand2 (2)+subtree (3) = 5
- **Option B: nand3 (3) = 3**



Part 2

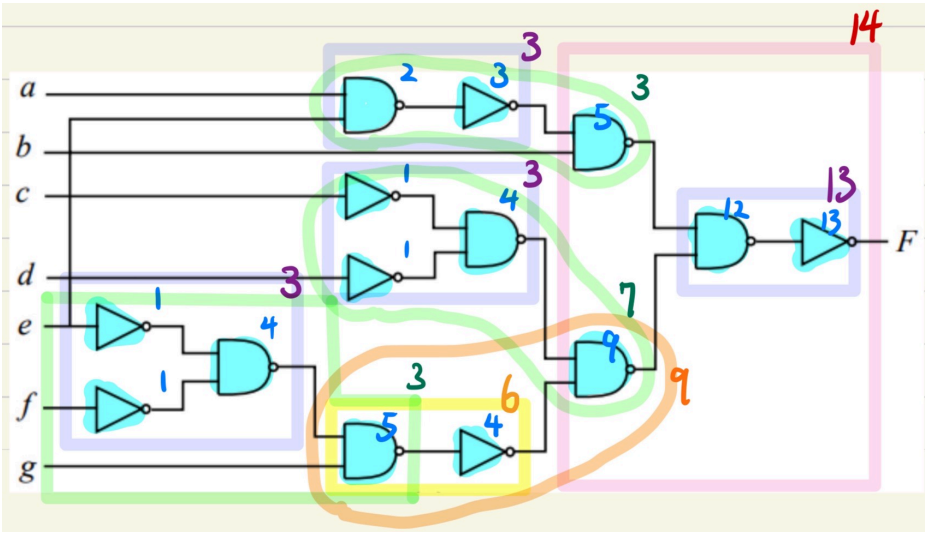
- Option A: nand2 (2)+subtree (3)+subtree (4) = 9
- **Option B: nor3 (3) +subtree (4) = 7**
- Option C: nand3 (3) +subtree (3)+subtree (3) = 9

6. **Optimal tree covering 6**



Optimal subtree covering cost = nand2(2) + subtree(3) + subtree(7) = 12

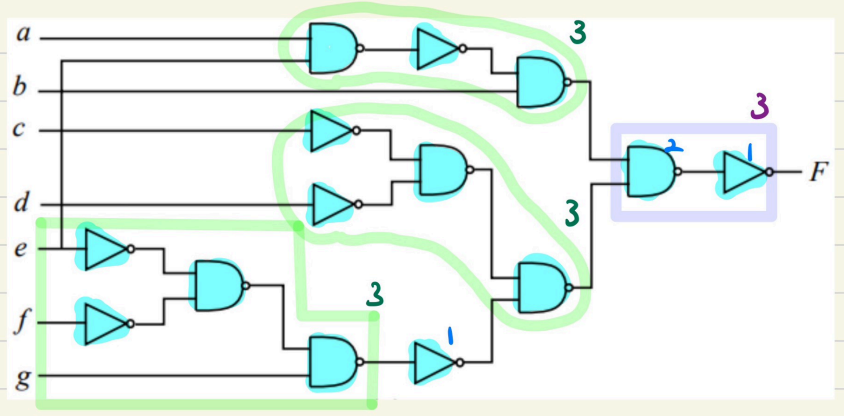
7. **Optimal tree covering 7**



Part 1

- **Option A: inv (1)+subtree (12) = 13**
- **Option B: and2 (3) +subtree (3) +subtree (7)= 13**
- Option C: aoi22 (4) +subtree (3)+subtree (3) +subtree (4)= 14

Answer



Using the cell library: **nor3 ×2 + inv x1 + nand3 ×1 + and2 ×1**

- t1 = nor3(e, f, g) = 3
- t2 = inv(t1) = 1
- t3 = nand3 (a, e, b) = 3
- t4 = nor3(c, d, t2) = 3
- t5 = and2(t3, t4) = 3

⇒ Optimal covering cost = 13

Bring up a BDD package

Operations & Functions

<整理>

分類	常用函式
初始化	`Cudd_Init` , `Cudd_Quit`
建立變數	`Cudd_bddNewVar` , `Cudd_ReadOne`
邏輯運算	`Cudd_bddAnd` , `Cudd_bddOr` , `Cudd_bddIte`
分析工具	`Cudd_Cofactor` , `Cudd_CountMinterm`
記憶體管理	`Cudd_Ref` , `Cudd_RecursiveDeref`
輸出	`Cudd_PrintDebug` , `Cudd_DumpDot`

<詳細使用方式與介紹>

初始化與終止	功能
`Cudd_Init()`	初始化一個 `DdManager` （必須第一步）
`Cudd_Quit(dd)`	釋放 BDD 管理器與所有資源

建立變數與常數	功能
`Cudd_bddNewVar(dd)`	建立新的布林變數
`Cudd_ReadOne(dd)`	取得常數 1 的節點
`Cudd_Not(node)`	回傳 node 的邏輯反（!node）

基本布林運算	功能
`Cudd_bddAnd(dd, f, g)`	f ∧ g
`Cudd_bddOr(dd, f, g)`	f ∨ g
`Cudd_bddXor(dd, f, g)`	f ⊕ g
`Cudd_bddIte(dd, i, t, e)`	ITE(i, t, e)：if i then t else e
`Cudd_bddNand(dd, f, g)`	非 AND
`Cudd_bddNor(dd, f, g)`	非 OR

函數操作與分析	功能
`Cudd_Cofactor(dd, f, cube)`	取得 f 在變數設定下的值（正/反向 cofactor）
`Cudd_Support(dd, f)`	取得 f 依賴的變數集合
`Cudd_CountMinterm(dd, f, nvars)`	計算 f 為真的輸入組合數（minterms）
`Cudd_bddExistAbstract(dd, f, cube)`	對某些變數做存在量化（∃）

記憶體管理	功能
`Cudd_Ref(node)`	增加節點引用數，避免被刪除
`Cudd_RecursiveDeref(dd, node)`	減少引用數，允許回收
`Cudd_CheckZeroRef(dd)`	檢查目前是否還有未釋放的節點

印出與視覺化	功能
`Cudd_PrintDebug(dd, f, nvars, level)`	印出 BDD 結構（console debug）
`Cudd_DumpDot(dd, n, farray, inames, onames, outfile)`	輸出成 Graphviz 的 .dot 檔

<自定義函式>

- `print_bdd_minterms`：輸出 minterms

```
void print_bdd_minterms(DdManager* dd, DdNode* f, int var_num) {
    printf("==== Minterms ====\n");
    DdGen *gen;
    int *inputs;
    CUDD_VALUE_TYPE val;

    Cudd_ForeachCube(dd, f, gen, inputs, val) {
        printf("f = 1 when: ");
        int i;
        for (i = 0; i < var_num; i++) {
            if (inputs[i] == 0)
                printf("x%d=0 ", i+1);
            else if (inputs[i] == 1)
                printf("x%d=1 ", i+1);
            else
                printf("x%d=? ", i+1); // don't care
        }
        printf("\n");
```

```
    }
}
```

`Cudd_ForeachCube(...)` 會走訪所有「為真」的輸入組合（即 minterms）。

- `export_bdd_to_dot` : 輸出.dot檔案 (BDD 結構)

```
void export_bdd_to_dot(DdManager* dd, DdNode* f, const char* filename, char** input_names, cl
    FILE *dotFile = fopen(filename, "w");
    if (!dotFile) {
        fprintf(stderr, "Error opening file %s for writing.\n", filename);
        return;
    }
    DdNode* farray[1] = {f};
    Cudd_DumpDot(dd, 1, farray, input_names, output_names, dotFile);
    fclose(dotFile);
    printf("\nDOT file '%s' generated.\n", filename);
}
```

DOT 檔的節點名稱會根據變數順序自動產生，傳 `inames[]` 可自定義輸出變數名稱。

- 呼叫方式 (in main function)

```
char* inames[3] = {"a", "b", "c"};
char* onames[1] = {"f"};
print_bdd_minterms(dd, f, var_num);
export_bdd_to_dot(dd, f, "bdd_f.dot", inames, onames);
```

生成 `bdd_f.dot` 後，可以用下列指令轉成圖檔來觀察 BDD 結構。

```
dot -Tpng bdd_case0.dot -o bdd_case0.png
```

Initialize

```
/* Initialize manager. We start with 0 variables */
  dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
  /*startCudd(option,net1->ninputs);*/
  if (dd == NULL) { exit(2); }

  /************ lets do our work here *************/
  one = Cudd_ReadOne( dd );
  Cudd_Ref(one); /* referenced for the first time */
  zero = Cudd_Not( one ); /* not the same as Cudd_ReadZero */
  /* Cudd_Ref(zero); */
  /* reference count increment not needed because Cudd_Not returns the projection */
```

每個CASE都需要重新 initialize

step 1. 初始化一個空的 `DdManager`，也就是 BDD 管理器。

`DdManager` 是 **CUDD 的中央管理物件**，負責管理：

- 所有的變數（如 a, b, c）
- 節點表（unique table）
- 快取記憶體（computed table）
- 節點共用與記憶體重用
- garbage collection（垃圾回收）

step 2. 建立 1 node 和 0 node (!1)

`Cudd_ReadOne(dd);`

從 BDD 管理器中取得「常數 1」節點，也就是代表「布林值為真」的終端節點。

- 這是個共享的、靜態存在的節點（全域唯一）
- 可以在任何地方安全地用它
- 如果要「長期使用」，就要 `Cudd_Ref(one);`

`Cudd_Ref(one);`

這代表你要告訴 CUDD：「我要長期使用這個節點，請不要幫我回收。」

CUDD 預設的 `one` 節點是存在於內部的，但如果你之後會在自己的運算中用它（例如建構 AND/OR），就應該加 `Cudd_Ref()` 來保留它。

`Cudd_Not(one);`

為何不是用 `Cudd_ReadZero()` ？

> ❗其實 CUDD 裡的 0 是用 Cudd_Not(one) 來實作的。

原因是：CUDD 使用了一種技巧：終端節點只實際存「1」，然後用 pointer flipping 來表示「非」。

也就是說：

- `Cudd_ReadOne()` 回傳的是終端節點 `1`
- `Cudd_Not(one)` 其實會在指標上翻轉一個位元（低位元做 XOR），來產生 `0`

這是為了節省記憶體與避免重複節點。

所以不應該用 `Cudd_ReadZero()` 來取得 0，而應該用 `Cudd_Not(Cudd_ReadOne())`。

CASE 0

Problem

https://my.ece.utah.edu/~kalla/ECE5740/main.c 上的範例程式，介紹 ITE(a, b, c) 用法

$$\text{if } (a) \text{ then } b; \text{ else } c \Rightarrow f = ab + \overline{a}c$$

Code

Part 1. 建立各個 `Ddnode` 之間的關係，並利用 `Cudd_PrintDebug` 輸出BDD的結構

```
/***** Add a new variable **********/
printf("\nCase 0: f = ab + a'c\n");
var_num = 3;
a = Cudd_bddNewVar(dd);
b = Cudd_bddNewVar(dd);
```

```
c = Cudd_bddNewVar(dd);
f = Cudd_bddIte(dd, a, b, c); /* compute ITE(a, b, c); */
Cudd_Ref(f);
/* print BDD structure of f */
printf("Printing the BDD for f: ptr to the nodes, T & E children\n");
Cudd_PrintDebug(dd, f, 3, 3);
```

Part 2. `Cudd_Cofactor`的使用方法，並比較g(Cofactor method) 和 h(直接 b and c) 是否等價

```
f_a = Cudd_Cofactor(dd, f, a); Cudd_Ref(f_a);
/* f_a = +ve cofactor of f w.r.t. a: f_a = b */
f_abar = Cudd_Cofactor(dd, f, Cudd_Not(a) ); Cudd_Ref(f_abar);
/* f_abar = -ve cofactor of f w.r.t. a: f_abar = c */
g = Cudd_bddAnd(dd, f_a, f_abar); Cudd_Ref(g);
printf("Printing the BDD for g = b AND c\n");
Cudd_PrintDebug( dd, g, 3, 3);
h = Cudd_bddAnd(dd, b, c); Cudd_Ref(h);
if( h == g ){
  printf("Equivalence h = g means same DdNode pointers\n");
}
else{
  printf("Equivalence h = g does not mean same DdNode pointers? Something must be wrong\n")
}
```

Part 3. 輸出 minterms 和 .dot 圖檔

```
char* inames_0[3] = {"a", "b", "c"};
print_bdd_minterms(dd, f, var_num);
export_bdd_to_dot(dd, f, "bdd_case0.dot", inames_0, onames);
```

Result

```
Case 0: f = ab + a'c
Printing the BDD for f: ptr to the nodes, T & E children
: 4 nodes 1 leaves 4 minterms
ID =  0x57617   index = 0       T = 0x57615     E =  0x57616
ID =  0x57616   index = 2       T = 1           E = !1
ID =  0x57615   index = 1       T = 1           E = !1

Printing the BDD for g = b AND c
: 3 nodes 1 leaves 2 minterms
ID =  0x57619   index = 1       T = 0x57616     E = !1
ID =  0x57616   index = 2       T = 1           E = !1

Equivalence h = g means same DdNode pointers
==== Minterms ====
f = 1 when: x1=0 x2=? x3=1
f = 1 when: x1=1 x2=1 x3=?
```

Output Explanation

- `Cudd_PrintDebug` 函式原型：

```
int Cudd_PrintDebug(DdManager *manager, DdNode *f, int n, int pr);
```

- 參數說明：

參數	意義
`manager`	BDD 管理器（`DdManager *`）
`f`	要印出的布林函數（`DdNode *`）
`n`	**最多顯示的變數數量**（用來限制顯示內容的範圍）
`pr`	**印出等級（verbosity）**，常用為 0～3，數字越大印越多細節

- 典型的 `pr` 值說明（print level）：

`pr` 值	印出內容
0	幾乎不印
1	印出基本節點資訊
2	多印一些子節點
3	印出所有節點的詳細 ID、變數 index、T/E 子節點、minterms 等等

- 舉例來說: `Cudd_PrintDebug(dd, f, 3, 3)` 的意思是
  - **列印函數 `f` 的 BDD 結構**
  - 顯示 **最多 3 個變數**
  - 使用 **詳細程度為 3 的印出格式**

part 1

```
Printing the BDD for f: ptr to the nodes, T & E children
: 4 nodes 1 leaves 4 minterms
ID =  0xafe97   index = 0      T = 0xafe95    E =  0xafe96
ID =  0xafe96   index = 2      T = 1          E = !1
ID =  0xafe95   index = 1      T = 1          E = !1
```

輸出 **布林函數 f 的 BDD 結構**，細節如下：

項目	說明
`ID`	該節點的指標（記憶體位址），如 `0xafe97`。
`index`	該節點代表的變數編號（變數 x_0 表示 index=0）。
`T`	Then child（變數為 1 時走的分支）。
`E`	Else child（變數為 0 時走的分支）。
`1` / `!1`	1 表示常數 1，`!1` 表示常數 0（即 CUDD 的 `Cudd_Not(1)`）。

所以：

- `0xafe97` 是 root node，代表變數 `x0`。

- 當 `x0 = 1` 時走 `T = 0xafe95` → 變數 `x1`。
- 當 `x0 = 0` 時走 `E = 0xafe96` → 變數 `x2`。

- `0xafe95` 是變數 `x1` 的節點（index=1），其 `T=1`, `E=!1` → 表示 `x1`
- `0xafe96` 是變數 `x2` 的節點（index=2），其 `T=1`, `E=!1` → 表示 `x2`

4 nodes 1 leaves 4 minterms:

> 函數 f 是一個三變數布林函數，只有 4 個 minterms（也就是只有 4 組輸入為真）。

項目	意義
nodes	BDD 內部節點的數量（非終端）
leaves	終端節點的數量，幾乎永遠是 `1`（即 One/True）
minterms	有多少輸入組合會使函數值為 `1`（真）

part 2

```
Printing the BDD for g = b AND c
: 3 nodes 1 leaves 2 minterms
ID =  0xafe99   index = 1      T = 0xafe96     E = !1
ID =  0xafe96   index = 2      T = 1           E = !1
```

這是另一個布林函數 `g = b ∧ c`（假設 `b = x1`, `c = x2`），可以對照上方：

- `0xafe99`: 變數 x1
  - `T=0xafe96`（也就是 x2 的節點）
  - `E=!1` → 如果 x1=0，結果是 0
- `0xafe96`: 變數 x2（同上）
  - `T=1` → x2=1 時為 1
  - `E=!1` → x2=0 時為 0

結果很直觀就是 x1 AND x2

part 3

```
Equivalence h = g means same DdNode pointers
```

這句意思是：

> h = g（也就是 h 是由 f 做某些操作得到的），最終結果與 g 邏輯上完全等價。
> 因為它們的根節點指標是相同的
> `DdNode` pointer，代表 BDD 被 CUDD 共用、合併了。

✅ 這是 CUDD 的一個關鍵優點：「相同函數會自動共用節點」，提升效能與記憶體使用效率。

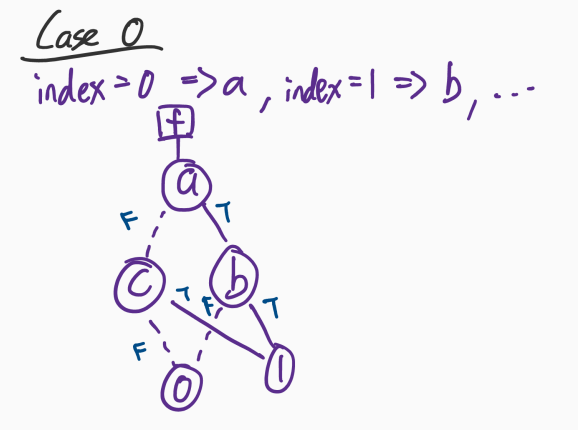
part 4

```
==== Minterms ====
f = 1 when: x1=0 x2=? x3=1
f = 1 when: x1=1 ×2=1 ×3=?
```

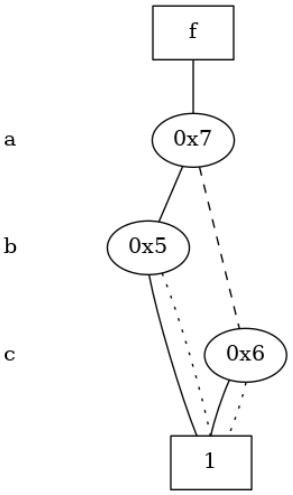
其中 ? 代表 don't care

BDD

依照 `Cudd_PrintDebug` 的結果進行手繪



使用 `Cudd_DumpDot`



為什麼 DOT 圖沒有顯示節點 "0"？

在 CUDD 的 `Cudd_DumpDot` 輸出中，**常數節點 `0`（false）** 是**預設會被省略的**，如果沒有任何 BDD 路徑連到它，或者它被某些選項過濾掉，**就不會畫出來**。這張圖只有終端節點 `1` 被連結，顯示的是哪些輸入組合會讓輸出為 `1`，也就是函數的「真值條件路徑」。

CASE 1

Problem

$$f = ab + ac + bc$$

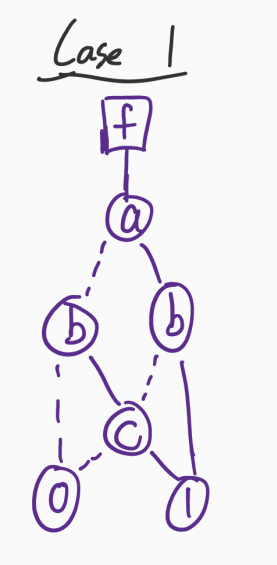
Code

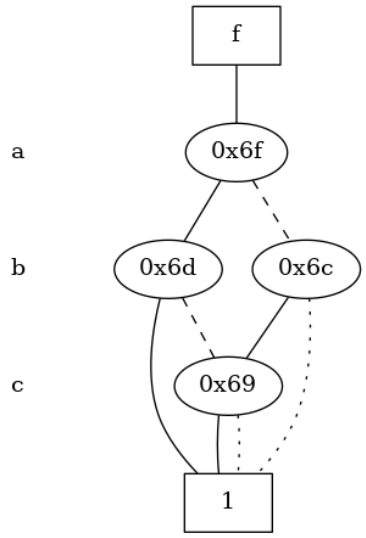
```
a = Cudd_bddNewVar(dd1);
b = Cudd_bddNewVar(dd1);
c = Cudd_bddNewVar(dd1);
ab = Cudd_bddAnd(dd1, a, b); Cudd_Ref(ab);
ac = Cudd_bddAnd(dd1, a, c); Cudd_Ref(ac);
bc = Cudd_bddAnd(dd1, b, c); Cudd_Ref(bc);
tmp = Cudd_bddOr(dd1, ab, ac); Cudd_Ref(tmp);
f = Cudd_bddOr(dd1, tmp, bc); Cudd_Ref(f);
Cudd_PrintDebug(dd1, f, var_num, 3);
```

Result

```
Case 1: f = ab + ac + bc
: 5 nodes 1 leaves 4 minterms
ID =  0x57b6f   index = 0       T = 0x57b6d     E =  0x57b6c
ID =  0x57b6c   index = 1       T = 0x57b69     E = !1
ID =  0x57b69   index = 2       T = 1           E = !1
ID =  0x57b6d   index = 1       T = 1           E =  0x57b69

==== Minterms ====
f = 1 when: x1=0 x2=1 x3=1
f = 1 when: x1=1 x2=0 x3=1
f = 1 when: x1=1 x2=1 x3=?
```

BDD





CASE 2

Problem

w	x	y	z	f
1	1	0	-	1
1	-	0	1	1
1	-	-	-	1
-	1	-	1	1
0	1	1	-	1
0	0	-	-	1
0	0	1	0	1

Code

```
printf("\nCase 2: Given a truth table, convert it into a BDD\n");
printf("Method 1: Sum of Product\n");
```

```
w = Cudd_bddNewVar(dd2);
x = Cudd_bddNewVar(dd2);
y = Cudd_bddNewVar(dd2);
z = Cudd_bddNewVar(dd2);

DdNode* vars[4] = {w, x, y, z};
DdNode* cubes[7];

int cube[7][4] = {
  {1, 1, 0, 2}, // 第一列: w=1 x=1 y=0 z=don't care
  {1, 2, 0, 1}, // 第二列: w=1, x=don't care, y=0, z=1
  {1, 2, 2, 2},
  {2, 1, 2, 1},
  {0, 1, 1, 2},
  {0, 0, 2, 2},
  {0, 0, 1, 0}
};
for(i = 0; i < 7; i++){
  cubes[i] = Cudd_bddComputeCube(dd2, vars, cube[i], 4); Cudd_Ref(cubes[i]);
}

// 把所有 c1 ~ c7 做 OR
f = zero;
for( i = 0; i < 7; i++){
  tmp  = Cudd_bddOr(dd2, f, cubes[i]); Cudd_Ref(f);
  Cudd_RecursiveDeref(dd2, f);
  f = tmp;
}
Cudd_PrintDebug(dd2, f, var_num, 3);
char* inames_2[4] = {"w", "x", "y", "z"};
print_bdd_minterms(dd2, f, var_num);
export_bdd_to_dot(dd2, f, "bdd_case2.dot", inames_2, onames);


// method 2
printf("Method 2: f = w + x' + y + z\n");
dd2 = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
if (dd2 == NULL) { exit(2); }
one = Cudd_ReadOne(dd2); Cudd_Ref(one);
zero = Cudd_Not( one );

w = Cudd_bddNewVar(dd2);
x = Cudd_bddNewVar(dd2);
y = Cudd_bddNewVar(dd2);
z = Cudd_bddNewVar(dd2);
tmp = Cudd_bddOr(dd2, w, Cudd_Not(x)); Cudd_Ref(tmp);
tmp = Cudd_bddOr(dd2, tmp, z); Cudd_Ref(tmp);
f = Cudd_bddOr(dd2, tmp, y); Cudd_Ref(f);
```

```
Cudd_PrintDebug(dd2, f, var_num, 3);

print_bdd_minterms(dd2, f, var_num);
export_bdd_to_dot(dd2, f, "bdd_case2_.dot", inames_2, onames);
```

方法1：SOP

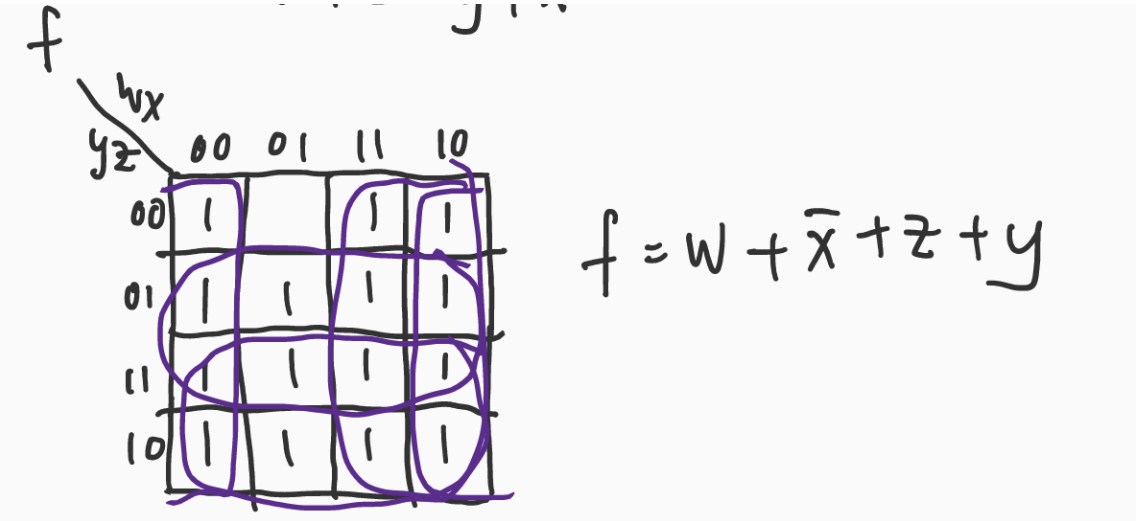
- `vars` 是變數陣列 `[w, x, y, z]`
- `cube[i]` 的值意義：
  - `0`: xi = 0
  - `1`: xi = 1
  - `2`: don't care

```
DdNode * Cudd_bddComputeCube(DdManager *dd, DdNode **vars, int *phase, int n)
```

產生一個符合變數條件的 BDD「乘積項（cube）」，常用來建立 minterms 或部分輸入的 pattern

方法二: 化簡 Truth table

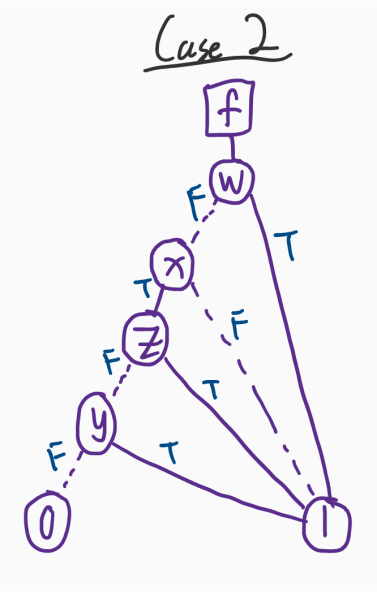
$$
\begin{aligned}
f &= w \cdot x \cdot \overline{y} + w \cdot \overline{y} \cdot z + w + x \cdot z + \overline{w} \cdot x \cdot y + \overline{w} \cdot \overline{x} + \overline{w} \cdot \overline{x} \cdot y \cdot \overline{z} \\
&= w + xz + xy + \overline{x} \\
&= w + \overline{x} + z + y
\end{aligned}
$$

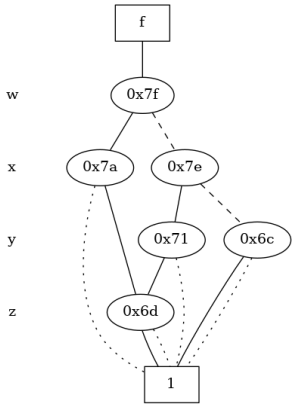


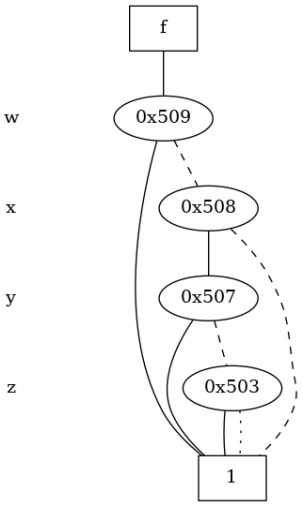
Result

```
Case 2: Given a truth table, convert it into a BDD
Method 1: Sum of Product
: 7 nodes 1 leaves 5 minterms
ID =  0xbe77f   index = 0       T = 0xbe77a     E =  0xbe77e
ID =  0xbe77e   index = 1       T = 0xbe771     E =  0xbe76c
ID =  0xbe76c   index = 2       T = 1           E = !1
ID =  0xbe771   index = 2       T = 0xbe76d     E = !1
ID =  0xbe76d   index = 3       T = 1           E = !1
ID =  0xbe77a   index = 1       T = 0xbe76d     E = !1

==== Minterms ====
f = 1 when: x1=0 x2=0 x3=1 x4=?
f = 1 when: x1=0 x2=1 x3=1 x4=1
f = 1 when: x1=1 x2=1 x3=? x4=1

DOT file 'bdd_case2.dot' generated.
Method 2: f = w + x' + y + z
: 5 nodes 1 leaves 15 minterms
ID =  0x12ed09  index = 0       T = 1           E =  0x12ed08
ID =  0x12ed08  index = 1       T = 0x12ed07    E =  1
ID =  0x12ed07  index = 2       T = 1           E =  0x12ed03
ID =  0x12ed03  index = 3       T = 1           E = !1

==== Minterms ====
f = 1 when: x1=0 x2=0 x3=? x4=?
f = 1 when: x1=0 x2=1 x3=0 x4=1
f = 1 when: x1=0 x2=1 x3=1 x4=?
f = 1 when: x1=1 x2=? x3=? x4=?
```

BDD







※ method 1的結果是錯的，原因仍然在排查

CASE 3

Problem

$$f = ab + cd + ef$$

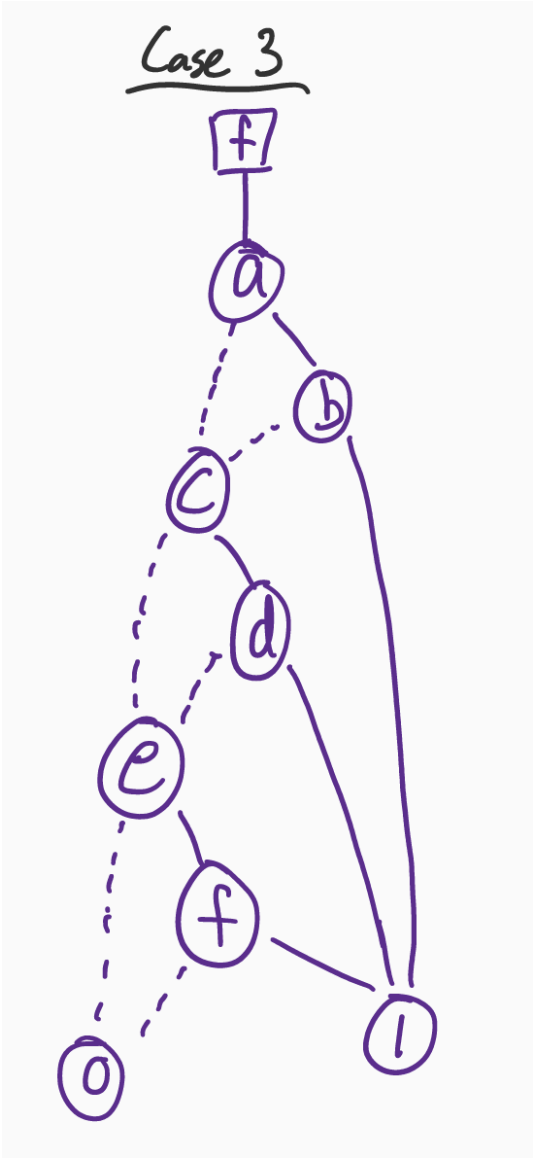
Code

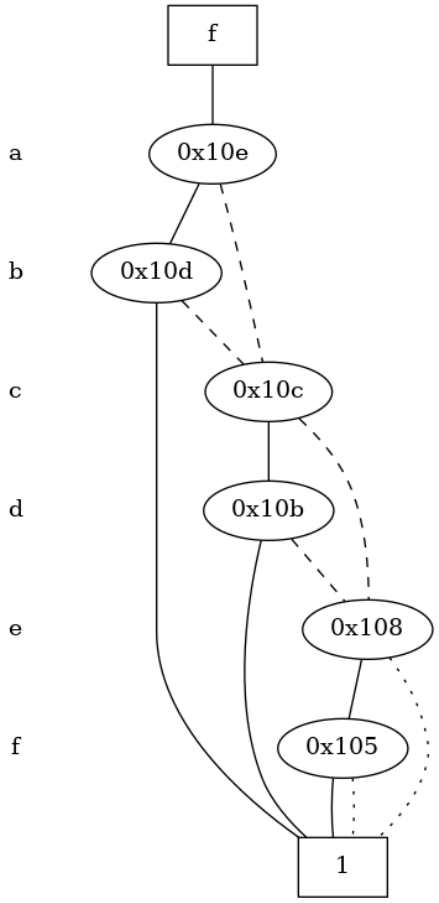
```
a = Cudd_bddNewVar(dd3);
b = Cudd_bddNewVar(dd3);
c = Cudd_bddNewVar(dd3);
d = Cudd_bddNewVar(dd3);
e = Cudd_bddNewVar(dd3);
f = Cudd_bddNewVar(dd3);
ab = Cudd_bddAnd(dd3, a, b); Cudd_Ref(ab);
cd = Cudd_bddAnd(dd3, c, d); Cudd_Ref(cd);
ef = Cudd_bddAnd(dd3, e, f); Cudd_Ref(ef);
tmp = Cudd_bddOr(dd3, ab, cd); Cudd_Ref(tmp);
f = Cudd_bddOr(dd3, tmp, ef); Cudd_Ref(f);
Cudd_PrintDebug(dd3, f, var_num, 3);
```

Result

```
Case 3: f = ab + cd + ef
: 7 nodes 1 leaves 37 minterms
ID =  0x13810e  index = 0       T = 0x13810d    E =  0x13810c
ID =  0x13810c  index = 2       T = 0x13810b    E =  0x138108
ID =  0x138108  index = 4       T = 0x138105    E = !1
ID =  0x138105  index = 5       T = 1           E = !1
ID =  0x13810b  index = 3       T = 1           E =  0x138108
ID =  0x13810d  index = 1       T = 1           E =  0x13810c

==== Minterms ====
f = 1 when: x1=0 x2=? x3=0 x4=? x5=1 x6=1
f = 1 when: x1=0 x2=? x3=1 x4=0 x5=1 x6=1
f = 1 when: x1=0 x2=? x3=1 x4=1 x5=? x6=?
f = 1 when: x1=1 x2=0 x3=0 x4=? x5=1 x6=1
f = 1 when: x1=1 x2=0 x3=1 x4=0 x5=1 x6=1
f = 1 when: x1=1 x2=0 x3=1 x4=1 x5=? x6=?
f = 1 when: x1=1 x2=1 x3=? x4=? x5=? x6=?
```

BDD





CASE 4

Problem

$$f = \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3} + \overline{x_1}\overline{x_2}x_3 + x_1\overline{x_2}\overline{x_3} + x_1x_2\overline{x_3} + x_1x_2x_3$$

Shannon expansion：

$$\begin{aligned} f_{x_1} &= \overline{x_2}x_3 + x_2\overline{x_3} + x_2x_3 \\ &\Rightarrow f_{x_1,x_2} = \overline{x_3} + x_3 = 1 \\ &\Rightarrow f_{x_1,\overline{x_2}} = x_3 \end{aligned}$$

$$\begin{aligned} f_{\overline{x_1}} &= \overline{x_2}\,\overline{x_3} + x_2\overline{x_3} + \overline{x_2}x_3 \\ &\Rightarrow f_{\overline{x_1},x_2} = \overline{x_3} \\ &\Rightarrow f_{\overline{x_1},\overline{x_2}} = \overline{x_3} + x_3 = 1 \end{aligned}$$

Code

Method 1: 使用 `Cudd_bddAnd` 和 `Cudd_bddOr` 把 Boolean function 建立起來

```
// method 1
printf("Method 1: Use AND OR combination\n");
tmp = Cudd_bddAnd(dd4, Cudd_Not(x1), Cudd_Not(x2)); Cudd_Ref(tmp);
a = Cudd_bddAnd(dd4, tmp, Cudd_Not(x3)); Cudd_Ref(a);
tmp = Cudd_bddAnd(dd4, Cudd_Not(x1), x2); Cudd_Ref(tmp);
b = Cudd_bddAnd(dd4, tmp, Cudd_Not(x3)); Cudd_Ref(b);
tmp = Cudd_bddAnd(dd4, Cudd_Not(x1), Cudd_Not(x2)); Cudd_Ref(tmp);
c = Cudd_bddAnd(dd4, tmp, x3); Cudd_Ref(c);
tmp = Cudd_bddAnd(dd4, x1, Cudd_Not(x2)); Cudd_Ref(tmp);
d = Cudd_bddAnd(dd4, tmp, x3); Cudd_Ref(d);
tmp = Cudd_bddAnd(dd4, x1, x2); Cudd_Ref(tmp);
e = Cudd_bddAnd(dd4, tmp, Cudd_Not(x3)); Cudd_Ref(e);
tmp = Cudd_bddAnd(dd4, x1, x2); Cudd_Ref(tmp);
g = Cudd_bddAnd(dd4, tmp, x3); Cudd_Ref(g);

tmp = Cudd_bddOr(dd4, a, b); Cudd_Ref(tmp);
tmp = Cudd_bddOr(dd4, tmp, c); Cudd_Ref(tmp);
tmp = Cudd_bddOr(dd4, tmp, d); Cudd_Ref(tmp);
tmp = Cudd_bddOr(dd4, tmp, e); Cudd_Ref(tmp);
f_from_AND_OR = Cudd_bddOr(dd4, tmp, g); Cudd_Ref(f);
Cudd_PrintDebug(dd4, f_from_AND_OR, var_num, 3);
```

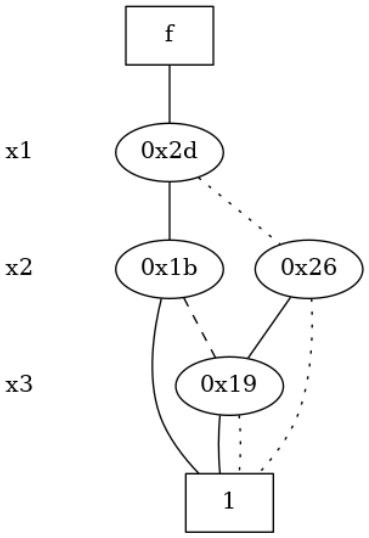
Method 2：使用 Shannon expansion 將方程式拆解後，使用 `Cudd_bddIte` 將其整合

```
printf("Method 2: ITE\n");
a = Cudd_bddIte(dd4, x2, one, x3); // if x1
b = Cudd_bddIte(dd4, x2, Cudd_Not(x3), one); // if x1'
f_from_ITE = Cudd_bddIte(dd4, x1, a, b);
Cudd_PrintDebug(dd4, f_from_ITE, var_num, 3);
```

Result

```
Case 4: f = x1'x2'x3' + x1'x2x3' + x1'x2'x3 + x1x2'x3 + x1x2x3' + x1x2x3
Method 1: Use AND OR combination
: 5 nodes 1 leaves 6 minterms
ID =  0x1a872d  index = 0       T = 0x1a871b    E = !0x1a8726
ID =  0x1a8726  index = 1       T = 0x1a8719    E = !1
ID =  0x1a8719  index = 2       T = 1           E = !1
ID =  0x1a871b  index = 1       T = 1           E =  0x1a8719

Method 2: ITE
: 5 nodes 1 leaves 6 minterms
ID =  0x1a872d  index = 0       T = 0x1a871b    E = !0x1a8726
ID =  0x1a8726  index = 1       T = 0x1a8719    E = !1
ID =  0x1a8719  index = 2       T = 1           E = !1
ID =  0x1a871b  index = 1       T = 1           E =  0x1a8719

The f generated by the two methods are logically equivalent!
==== Minterms ====
f = 1 when: x1=0 x2=0 x3=?
f = 1 when: x1=0 x2=1 x3=0
f = 1 when: x1=1 x2=0 x3=1
f = 1 when: x1=1 x2=1 x3=?
```

BDD



CASE 5

Problem

$$f = \overline{x_1 x_3} + \overline{x_2 x_3} + x_1 x_2$$

Shannon expansion：

$$
\begin{aligned}
f_{x_1} &= x_2 + \overline{x_2} x_3 \\
&\Rightarrow f_{x_1, x_2} = 1 \\
&\Rightarrow f_{x_1, \overline{x_2}} = x_3
\end{aligned}
$$

$$
\begin{aligned}
f_{\overline{x_1}} &= \overline{x_3} + \overline{x_2} x_3 \\
&\Rightarrow f_{\overline{x_1}, x_2} = \overline{x_3} \\
&\Rightarrow f_{\overline{x_1}, \overline{x_2}} = \overline{x_3} + x_3 = 1
\end{aligned}
$$

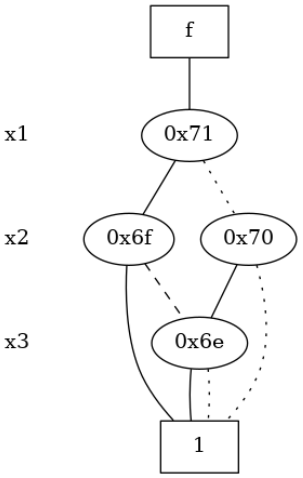
Code

```
x1 = Cudd_bddNewVar(dd5);
x2 = Cudd_bddNewVar(dd5);
x3 = Cudd_bddNewVar(dd5);

a = Cudd_bddIte(dd5, x2, one, x3); // if x1
b = Cudd_bddIte(dd5, x2, Cudd_Not(x3), one); // if x1'
f = Cudd_bddIte(dd5, x1, a, b);
Cudd_PrintDebug(dd5, f, var_num, 3);
```

Result

```
Case 5: f = x1'x3' + x2'x3 + x1x2
: 5 nodes 1 leaves 6 minterms
ID =  0x218c71  index = 0       T = 0x218c6f    E = !0x218c70
ID =  0x218c70  index = 1       T = 0x218c6e    E = !1
ID =  0x218c6e  index = 2       T = 1           E = !1
ID =  0x218c6f  index = 1       T = 1           E =  0x218c6e

==== Minterms ====
f = 1 when: x1=0 x2=0 x3=?
f = 1 when: x1=0 x2=1 x3=0
f = 1 when: x1=1 x2=0 x3=1
f = 1 when: x1=1 x2=1 x3=?
```

BDD



CASE 6

Problem

$$f = \overline{x_1}\overline{x_2} + x_2\overline{x_3} + x_1\overline{x_3}$$

Shannon expansion：

$$\begin{aligned} f_{x_1} &= x_3 + x_2\overline{x_3} \\ &\Rightarrow f_{x_1,x_2} = \overline{x_3} + x_3 = 1 \\ &\Rightarrow f_{x_1,\overline{x_2}} = x_3 \end{aligned}$$

$$\begin{aligned} f_{\overline{x_1}} &= \overline{x_2} + x_2\overline{x_3} \\ &\Rightarrow f_{\overline{x_1},x_2} = \overline{x_3} \\ &\Rightarrow f_{\overline{x_1},\overline{x_2}} = 1 \end{aligned}$$

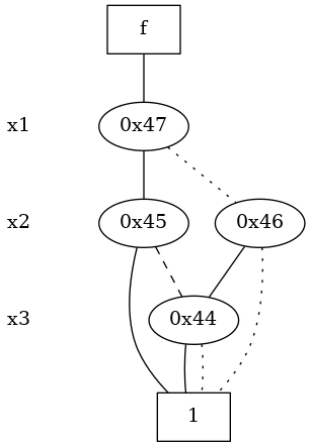
Code

```
x1 = Cudd_bddNewVar(dd6);
x2 = Cudd_bddNewVar(dd6);
x3 = Cudd_bddNewVar(dd6);

a = Cudd_bddIte(dd6, x2, one, x3); // if x1
b = Cudd_bddIte(dd6, x2, Cudd_Not(x3), one); // if x1'
f = Cudd_bddIte(dd6, x1, a, b);
Cudd_PrintDebug(dd6, f, var_num, 3);
```

Result

```
Case 6: f = x1'x2' + x2x3' + x1x3
: 5 nodes 1 leaves 6 minterms
ID =  0x2891c7  index = 0       T = 0x2891c5    E = !0x2891c6
ID =  0x2891c6  index = 1       T = 0x2891c4    E = !1
ID =  0x2891c4  index = 2       T = 1           E = !1
ID =  0x2891c5  index = 1       T = 1           E =  0x2891c4

==== Minterms ====
f = 1 when: x1=0 x2=0 x3=?
f = 1 when: x1=0 x2=1 x3=0
f = 1 when: x1=1 x2=0 x3=1
f = 1 when: x1=1 x2=1 x3=?
```

BDD



可以看到 case4~case6的`f`其實都是等價的