Quine-McCluskey
Report

13조

팀장: 2018202050 송영욱

조원: 2018202060 이준형

조원: 2018202048 이민재

Contents

1) Problem statements

Quine-McCluskey는 Karnaugh map과 달리 input 값의 수에 관계없이 Bool 표현식을 최소화하는 알고리즘입니다. 그리고 이 알고리즘을 우리가 배우고 있는 C++언어를 통해 구현하는 것이 문제입니다. 구현을 하여 연산 후 최종 결과의 Bool 표현식을 출력하고 활용해 여기에 사용되는 transistor의 수를 계산하여 Cost로 출력해 줍니다. 예를 들어 A'BC' + C'DEF는 3and gate (8)와 4and gate (10) 그리고 2or gate (6) 2개의 not gate (총4) 즉 28개의 transistor가 필요합니다. 파일 입출력을 사용하여 Input format으로 input minterm.txt을 받고 Output format으로 result.txt를 생성합니다.

예시)

Input format	**Output format**
File name : input_minterm.txt	File name : result.txt
4 // input bit length d 0000 // don't care value m 0100 // input having the result with true m 0101 m 0110 m 1001 m 1010 d 0111 d 1101 d 1111	01-- 1-01 1010 Cost (# of transistors): 40

Quine-McCluskey를 설명하자면 Column1에 있는 minterm들 중 hamming distance가 1인 minterm들을 찾아 다른 부분을 '-'로 대체하여 다음 Column2로 넘겨줍니다. 조합이 된minterm들은 체크해주고 조합이 되지 못해 체크가 되지 못한 mintem은 pls(prime implicants)가 됩니다. 이를 반복하여 최종 pls들과 true minterm들을 비교하여 essential pls를 선정합니다. essential pls로 모든 True minterm들이 커버되지 않는다면 추가로 최적의 pls를 선택해 커버해줍니다.

최종 선택된 pls들의 합에서 Cost 즉 만들 때 필요한 transistor의 수를 계산하여 result.txt에 저장합니다.

2-1) Psudo code

```
/*step 0 LOAD
ex)4//bitnum
m 1000 //true minterm
d 1001 //don't care*/
if("input_minterm.txt"==open())//txt파일이 열린경우
{    then,
     while(input_minterm.txt!=EOF)//txt파일이 끝날때까지
     {     function input{//txt파일에 있는것을 받아들입니다.
           getline(newminterm,100);
           if(first) then bitnum=newminterm->value;//첫 문자는 bitnum값
          newminterm->next;//문자열을 받을경우에는 다음 노드를 가리킵니다.
                          }
      }
}
```

```
//step 1 : make first column
cur=head;
while(cur=NULL)//input받은 minterm 처음부터 끝까지
{//trueminterm인지 don't care minterm인지 구별
    if(first spelling of minterm =='d')//첫 스펠링이 'd'이면 don't care minterm
         then minterm == don't care_minterm;
    else if(first spelling of minterm =='m')//첫 스펠링이 'm'이면 true minterm
         then minterm == true_minterm;
    char temp[100];
     int t = 0;
   for (int i = 2; i < bitnum + 2; i++)//i=2인 이유는 4_value형태이므로
       {
           temp[t] = minterm[i];
           minterm[t] = temp[t];
           t++;
       }
   minterm[bitnum] = '\0';
   cur->next;//다음 minterm 가르키기}
```

```
//step 2 : make another columns and, pls
    standard_minterm = head->next;
    while (standard_minterm != NULL)//기준노드 하나 잡고
    {
        cur_minterm = standard->next;
        while (cur != cutline->next)//standard랑 비교하기
        {
            for (int i = 0; i < bitnum; i++)
            {
                if (cur->value[i] == standard->value[i]) then  countcheck++;//해당 비트자리에 값이 같을경우
                else then differentpoint = i;//다를 경우 인덱스 저장
            }
            if (countcheck == bitnum - 1)//한 bit만 다른 경우
            {   then,
                strcpy_s(newminterm->value, standard->value);//새로운 Minterm 노드 생성, 값 입력
                newminterm->value[differentpoint] = '_';
                tail->next = newminterm;
                                //cutline의 next부터가 만들어 질 수 있는 column2,3, … N임
                tail = newminterm;
            }
            cur->next;//다음 minterm 가리키기
        }
        standard->next;//다음 기준 노드 가리키기
        if (standard == cutline)//각의 column 끝까지 확인할 때 마다
        {   then,
            standard->next;//can't combined인 경우 pls에 넣기
            cutline = tail;//cutline은 다음 colume의 끝가리키기
         }
     }
```

```
//step 2.5 : delete duplicated pls(중복 pls제거)
    standard = plshead;//standard노드는 plshead를 가리킴
    Minterm* mintermkiller;//지우개
    while (standard != NULL)
    {
        pls= standard->next;
        plspre= standard;
        while (pls != NULL)
        {
            if (standard==pls)
            {
                if (pls == standard->next)//첫번째가 중복이면
                {   then,
                    mintermkiller = plscur;
                    standard->next = plscur->next;
                    delete(mintermkiller);
                    pls = standard->next;
                }
                else if (pls == plstail)//맨 끝이 중복이면
                {   then,
                    plspre->next = NULL;
                    mintermkiller = pls;
                    delete(mintermkiller);
                    plstail = plspre;
                    break;
                }
                else//중간에 중복이면
                {   then,
                    plspre->next = pls->next;
                    mintermkiller = pls;
                    pls->next;
                    delete(mintermkiller);
                }
            }
            else
            {   then,
                plspre= pls;
                pls->next;//다음 pls를 가리킴
            }
        }
        standard->next;//다음 standard노드를 가리킴
```

```
//step 3 : true_minterm들 tlist에 넣기
    tlist;//true_minterm
    while (1)
    {
        if (minterm이 true minterm이면)
            then tlist.addtlist(minterm);//tlist에 true_minterm 넣기
        if (minterm == column1line)//column1까지만 확인
            then break;
        minterm->next;
    }
```

```
//step 4 : essential_pls를 찾기 위해 pls가 커버하는 수 체크
    while (true_minterm != NULL)
    {
        while (pls != NULL)
        {
            for (int i = 0; i < bitnum; i++)
            {
                if ((pls->value[i] == '_') || (pls->value[i] == tlist->tvalue[i]))
                {//don't care이거나 문자가 같을경우
                    if (i == bitnum - 1)
                    {
                        tlist->coverd++;//커버된 수 추가
                        pls->addtmin(tlist->tvalue);//pls가 커버한minterm 넣어주기
                        pls->countcover++;//커버한 수 추가
                    }
                }
                else then break;
            }
            pls->next;//다음 pls 가리키기
        }
        tlist->next;//다음 true_minterm 가리키기
    }
```

```
//step 5 : step4의 하나만 커버될 경우 그 경우를 essential_pls로 보냄
    while (tlistcur!= NULL)//하나만 커버되는 경우가 있다면 그 pls essentialpls
    {
        if (tlist->coverd == 1)//하나만 커버되는 거 찾고
        {
            while (pls != NULL)
            {
                if (tlist== NULL)
                    then goto finish;//다 커버 될 경우, finish로 간다.
                int find = pls->isthere(tlist->tvalue);//true_minterm를 그 pls가 cover하는지 체크
                if (find == 0)//이 민텀를 가진 pls찾았다면
                {   then, // essential pl 추가
                    newminterm = new Minterm;
                    strcpy_s(newminterm->value, plscur->value);
                    if (essentialhead == NULL)
                    {   then,
                        essentialhead = newminterm;
                        essentialtail = newminterm;
                    }
                    else
                    {   then,
                        essentialtail->next = newminterm;
                        essentialtail = newminterm;
                    }
                    plscur->cur = plscur->head;
                    while (plscur->cur != NULL)//이pls로커버된 민텀 체크
                    {
                        listcur2 = tlist.tlisthead;//listcur은 true_minterm
                        while (listcur2 != NULL)
                        {
                            if (strcmp(listcur2->tvalue, plscur->cur->tvalue) == 0)
                                then listcur2->check = 1;
                            listcur2->next;
                        }
                        //pls에서 커버된 민텀 체크해주기
                        strcpy_s(deletemin, plscur->cur->tvalue);
                        plscur3 = plshead;
                        while (plscur3 != NULL)
                        {
                            plscur3->checker(deletemin);
                            plscur3->next;//다음 노드
                        }
                        plscur->cur = plscur->cur->next;
                    }
                }
                pls->next;//다음 pls를 가리킴
            }
        }
        tlist->next;//다음 true_minterm를 가리킴
```

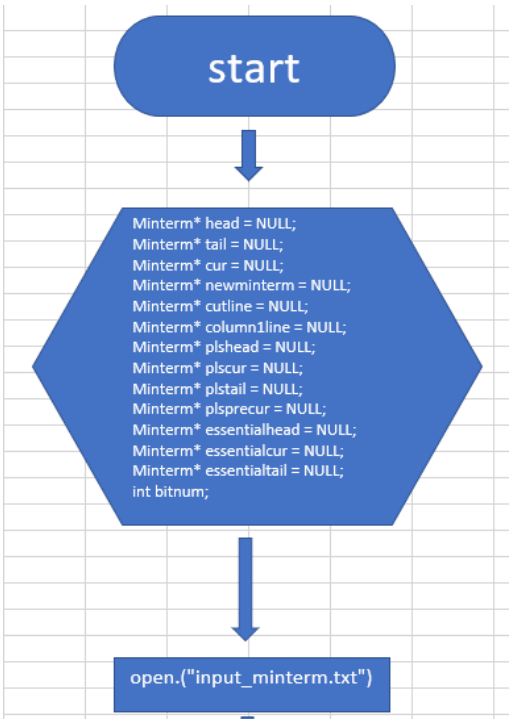
```
//step 7 : 트랜지스터의 개수 세기
    int checknot[100] = { 0, };
    essentialcur = essentialhead;
    while (essentialcur != NULL)
    {
        countandinput = 0;
        for (int i = 0; i < bitnum; i++)
        {
            if (essentialcur->value[i] == '0')
                then checknot[i] = 1;
            if ((essentialcur->value[i] == '0') || (essentialcur->value[i] == '1'))
                then countandinput++;
        }
        countand += countandinput * 2 + 2;
        countor++;
        essentialcur->next;
    }
    countor = countor * 2 + 2;
    for (int i = 0; i < bitnum; i++)
    {
        if (checknot[i] == 1)
        then countnot++;
    }
    countnot = 2 * countnot;
    cost = countnot + countand + countor; //총 cost
```

```
//step 8 : result.txt에 저장
    ofstream fout("result.txt");
    while (essentialcur != NULL)
    {
        write(essentialcur->value);
        essentialcur = essentialcur->next;
    }
    write(cost);
    //true_minterm을 커버하는 pls와 총 transistors의 갯수를 txt파일에 저장
```

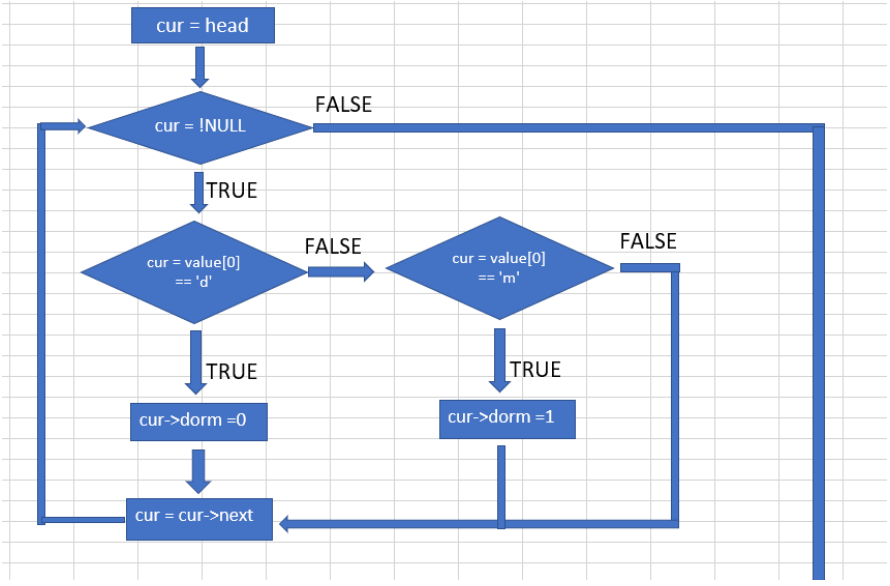
```
//step 9 : 동적할당 해제
    Tminterm* nextdeltminterm;
    Tminterm* deltminterm;
    deltminterm = true_minterm;
    while (deltminterm != NULL)
    {//True minterm들을 저장한 Tlist 삭제
        nextdeltminterm = deltminterm->next;
        delete(deltminterm);
        deltminterm = nextdeltminterm;
    }
    Minterm* nextdelminterm=NULL;
    Minterm* delminterm=head;
    while (delminterm != NULL)
    {//연산할 때 쓰였던 column 1, 2, 3 … 삭제
        nextdelminterm = delminterm->next;
        delete(delminterm);
        delminterm = nextdelminterm;
    }
    while (pls != NULL)
    {//pls에 달려있는 커버하는민텀들 삭제
        nextdeltminterm=NULL;
        deltminterm = pls->head;
        while (delminterm != NULL)
        {
            nextdeltminterm = deltminterm->next;
            delete(deltminterm);
            deltminterm = nextdeltminterm;
        }
         plscur->next;//다음 pls를 가리킴
    }
    nextdelminterm = NULL;
    delminterm = plshead;
    while (delminterm != NULL)
    {//pls list 삭제
        nextdelminterm = delminterm->next;
        delete(delminterm);
        delminterm = nextdelminterm;
    }
    nextdelminterm = NULL;
    delminterm = essentialhead;
    while (delminterm != NULL)
    {//essential list삭제
        nextdelminterm = delminterm->next;
        delete(delminterm);
        delminterm = nextdelminterm;
    }
```

2-2) Flow chart

<시작>

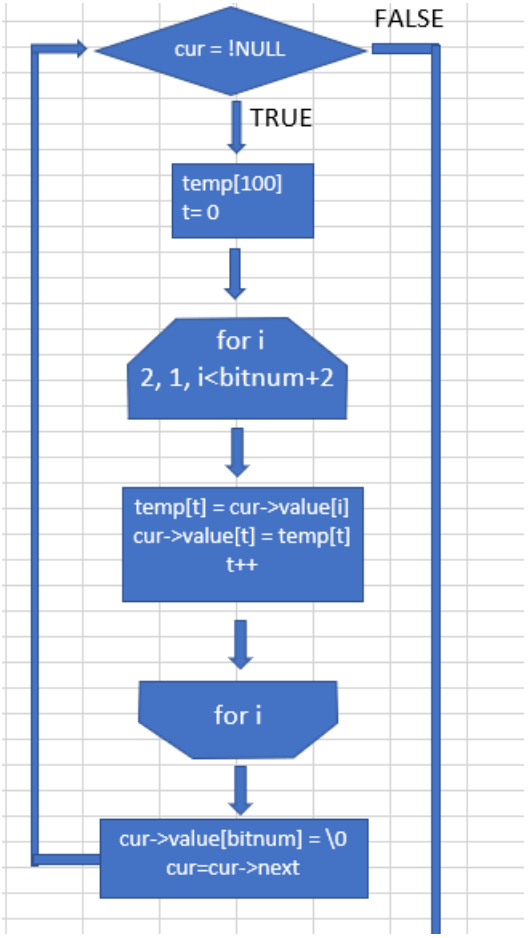


<d 랑 m 값을 dorm 에 저장>



만약 d 라면 dorm = 0 로 m 이라면 1 로 저장한다.

<d 랑 m 을뺀 순수 value 값만 저장>



Cur 을 움직여 가면서 d 랑 m 을 뺀 순수 value 값만 저장한다.

Ex) d 1001

m 1011

d 1111

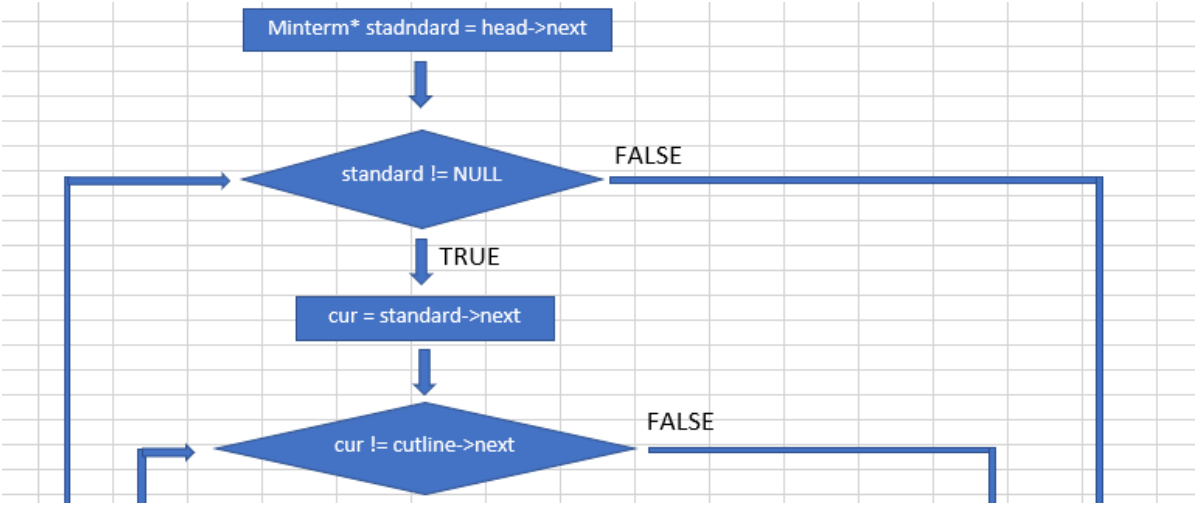
값이 있었다면

1001

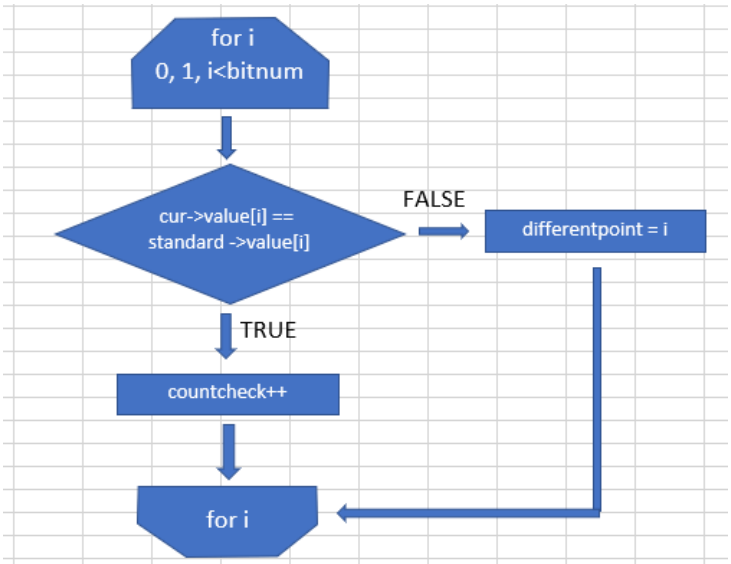
1011

1111 로 저장한다.

<standard 와 cur 을 비교>

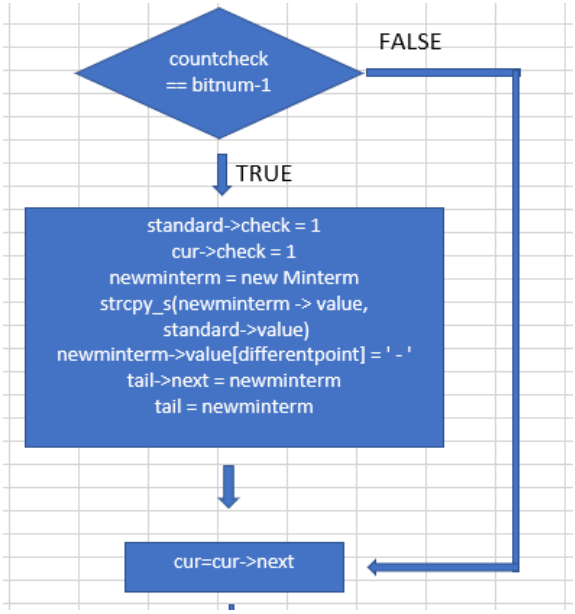


<value 의 각 자리 비교>



for 문으로 반복하면서 value 의 각 자리의 값을 standard 의 각 자리와 비교하면서 다른 값이 있다면 FALSE 로 그때의 자리를 differentpoint 에 저장한다

<한 bit 만 다를 경우>

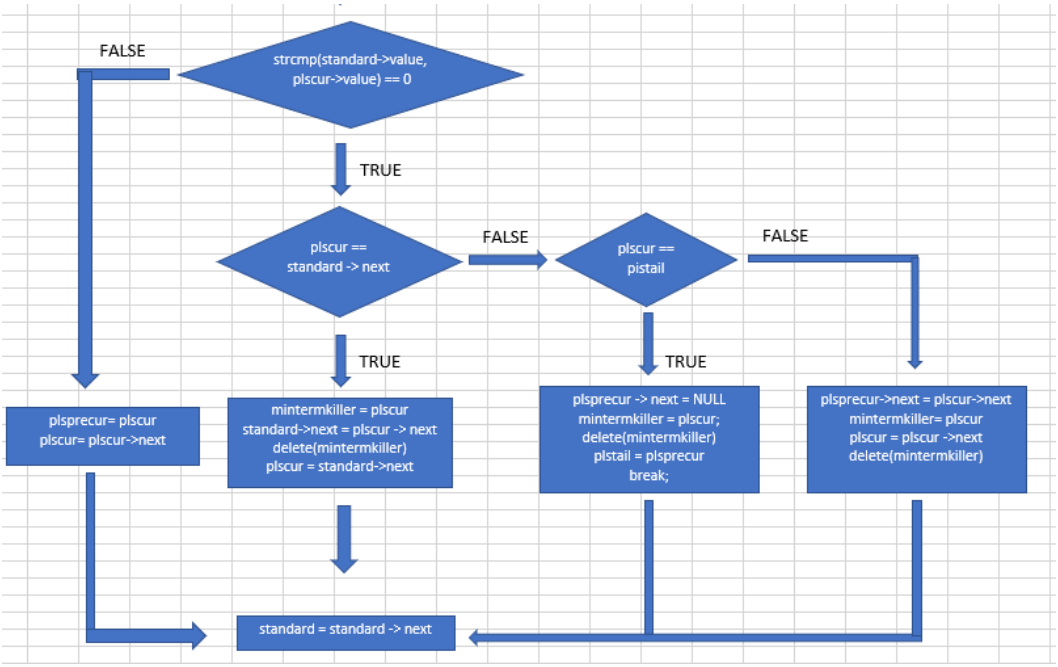


한 bit 만 다를 경우 그때의 standard 와 cur 의 check 값을 1 로 바꿔주고

새로운 node 를 만들어 값을 저장한다

저장된 differentpoint 의 값이 있는 위치를 '-'로 바꿔준다.

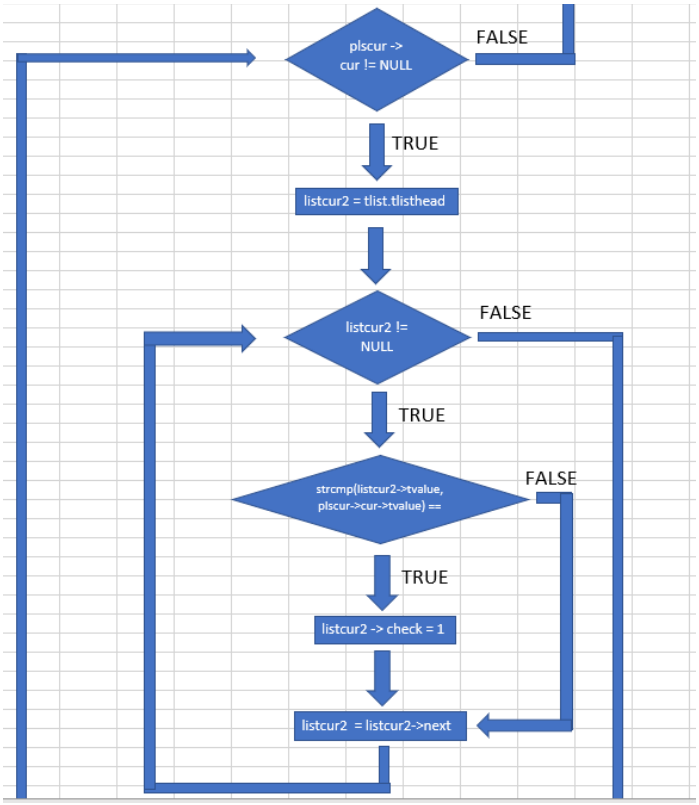
<중복되는 pls 제거>



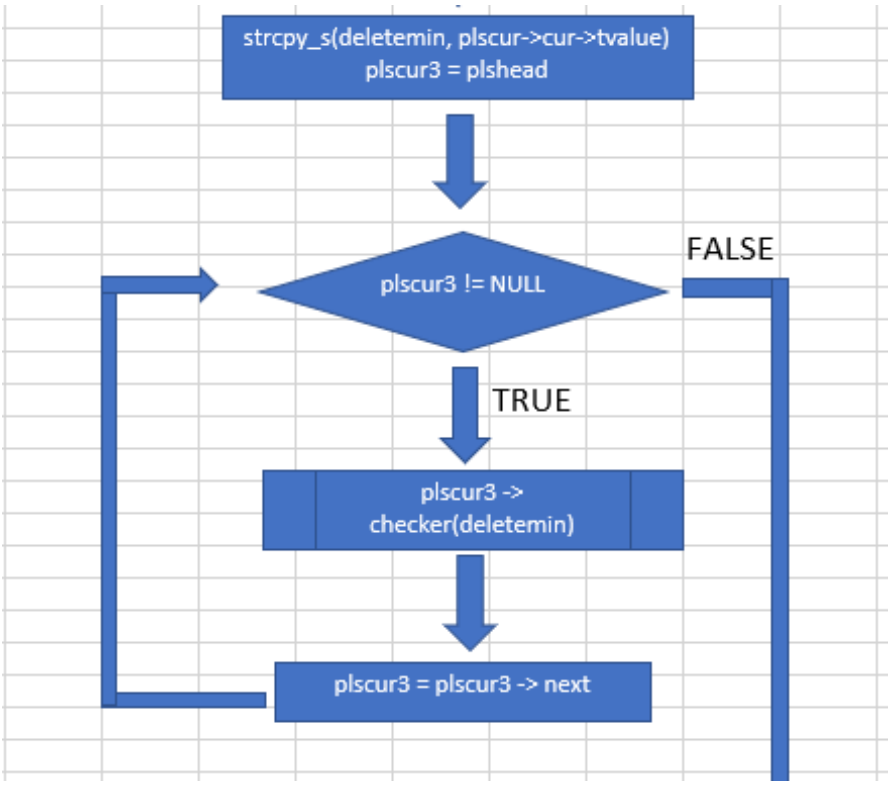
중복되는 pls 제거는 경우를 나누어서 제거하였습니다.

1. plscur 가 standard 와 붙어있을경우
2. plscur 가 plstail 과 같은경우
3. 그 외의 경우

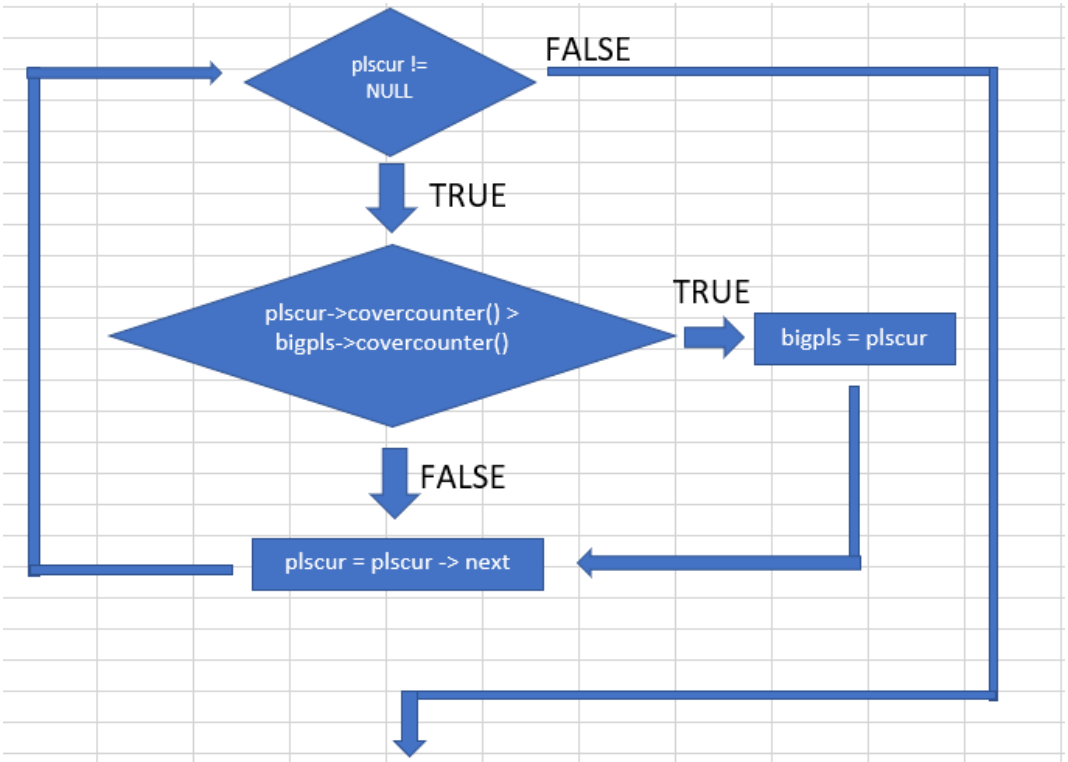
<essential pls 가 cover 하는 minterm 들 체크>



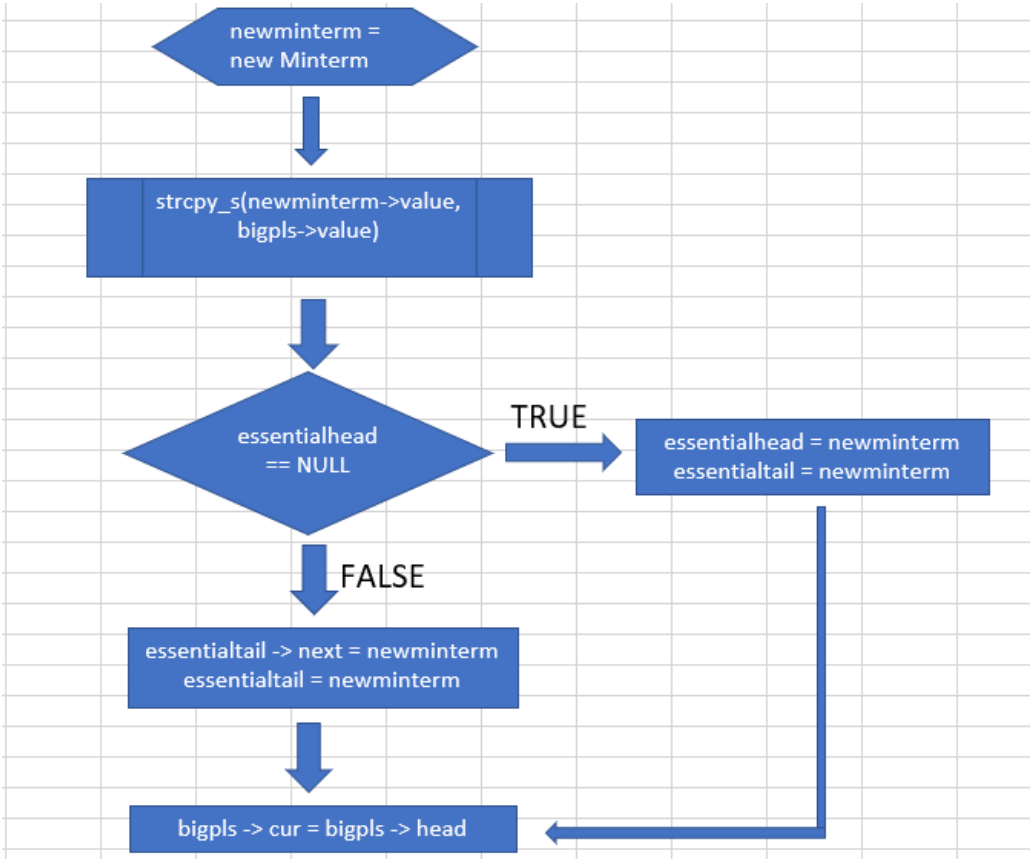
<pls 에서 cover 된 minterm 들 체크>



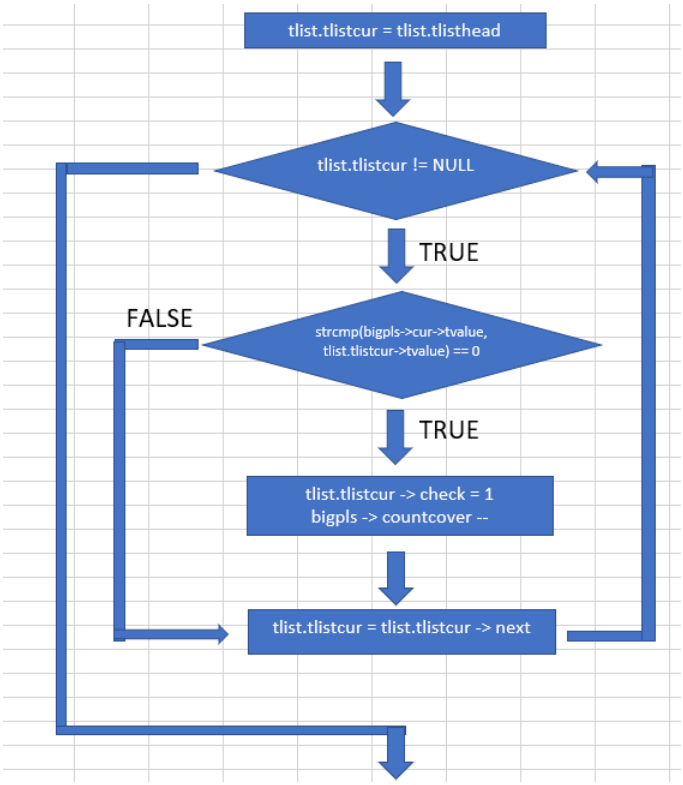
<남은 minterm 들중 가장 많이 cover 하는 pls 선택>



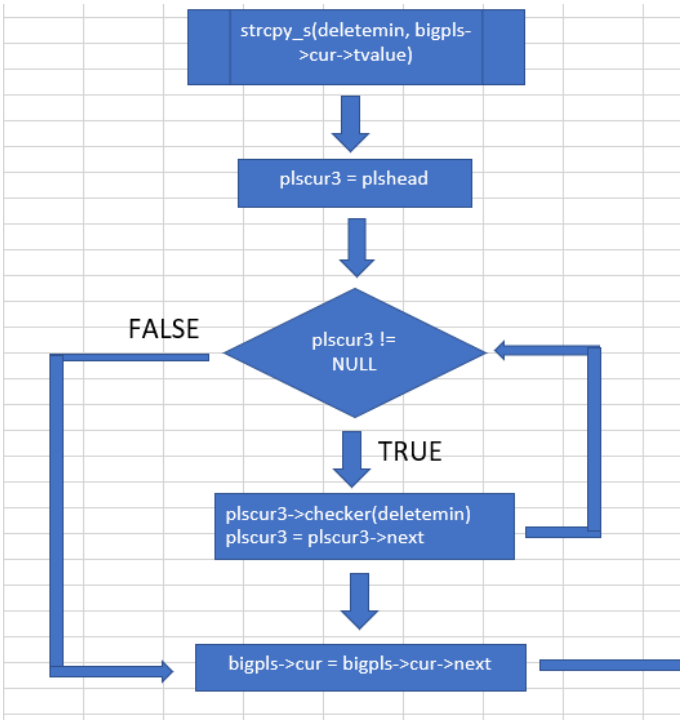
<가장 cover 많이하는 pls 를 에센셜로 보내기>



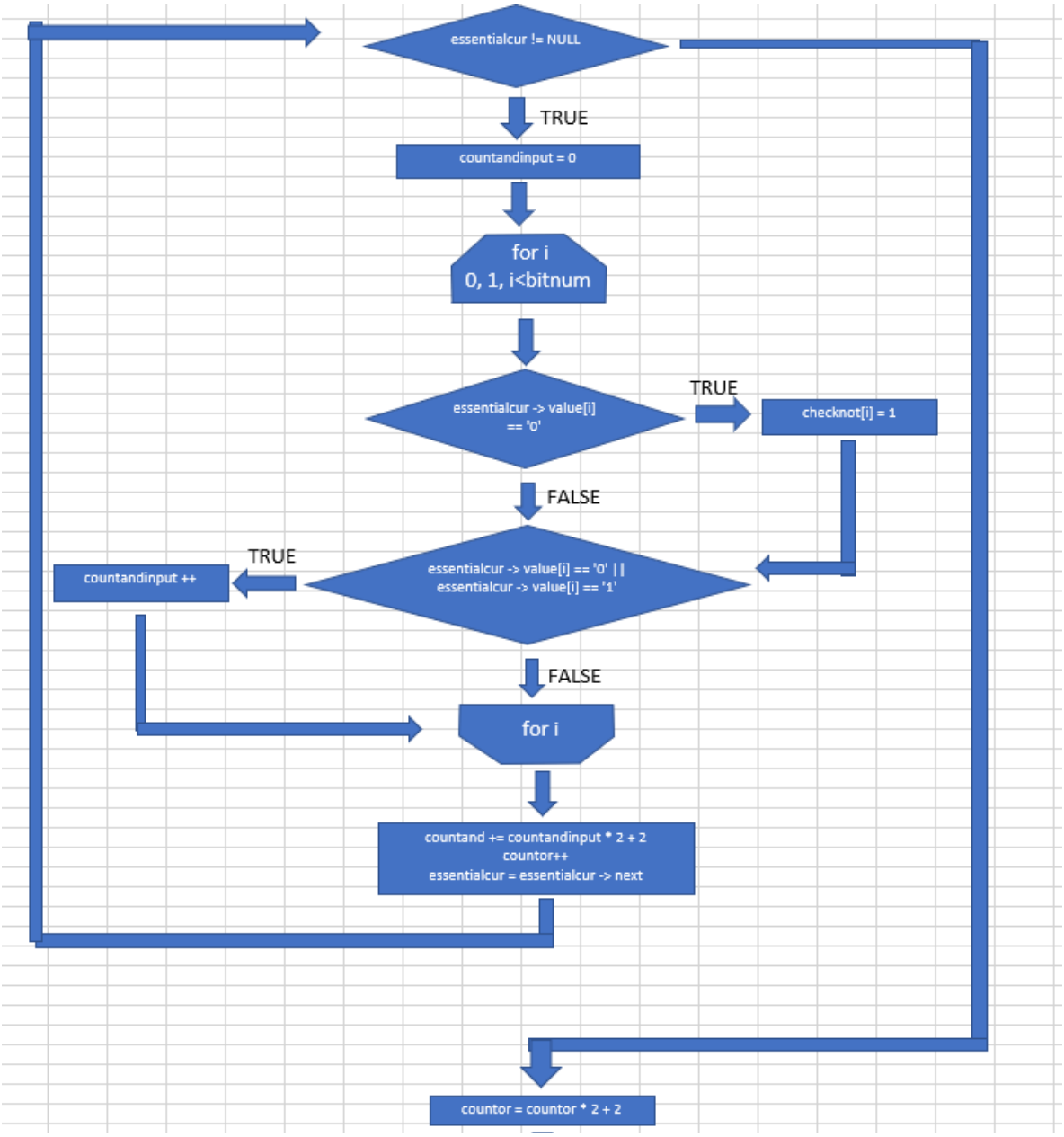
<trueminterm에서 cover된 minterm 체크>

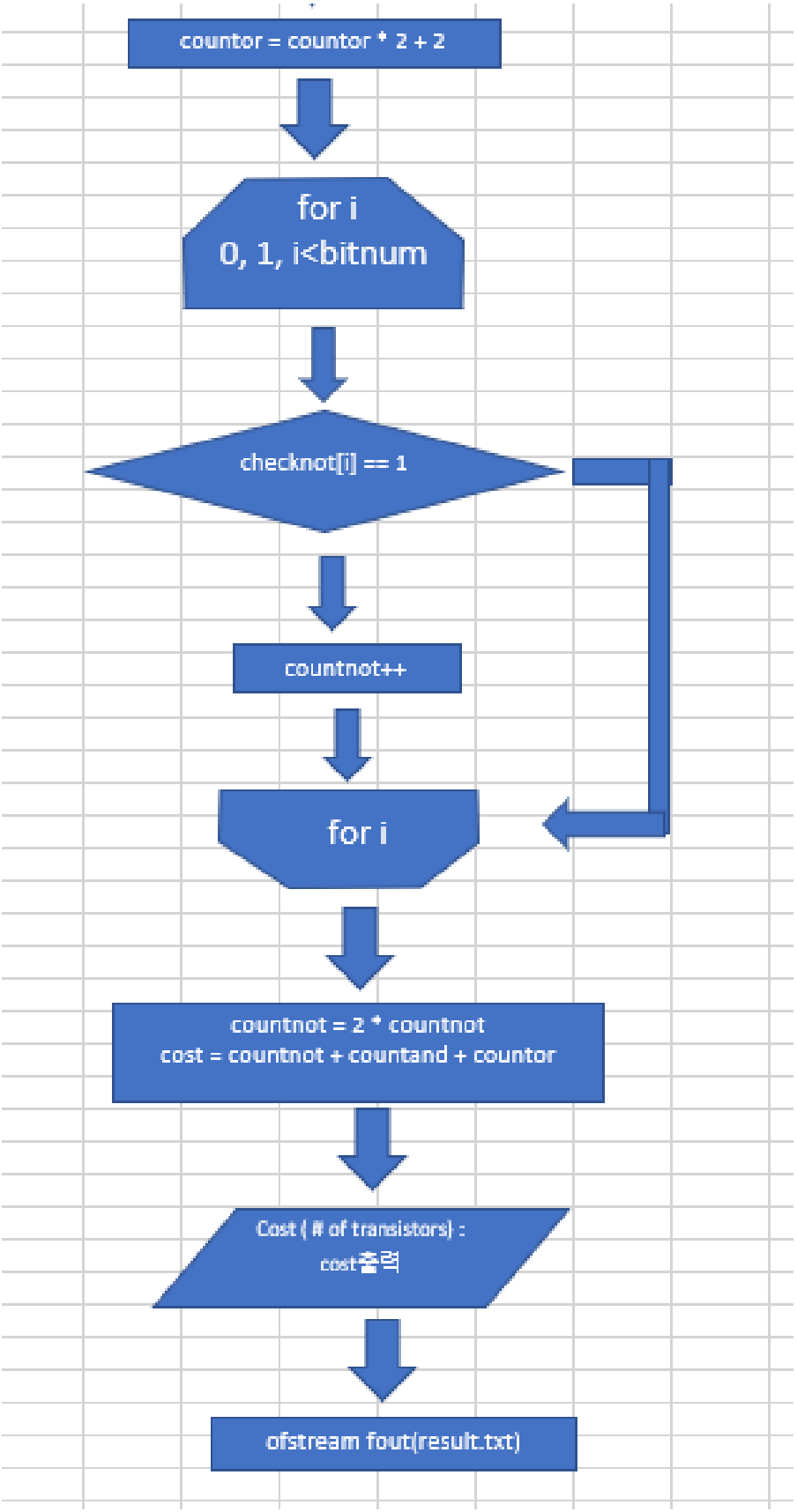


<pls에서 커버된 minterm들 체크>

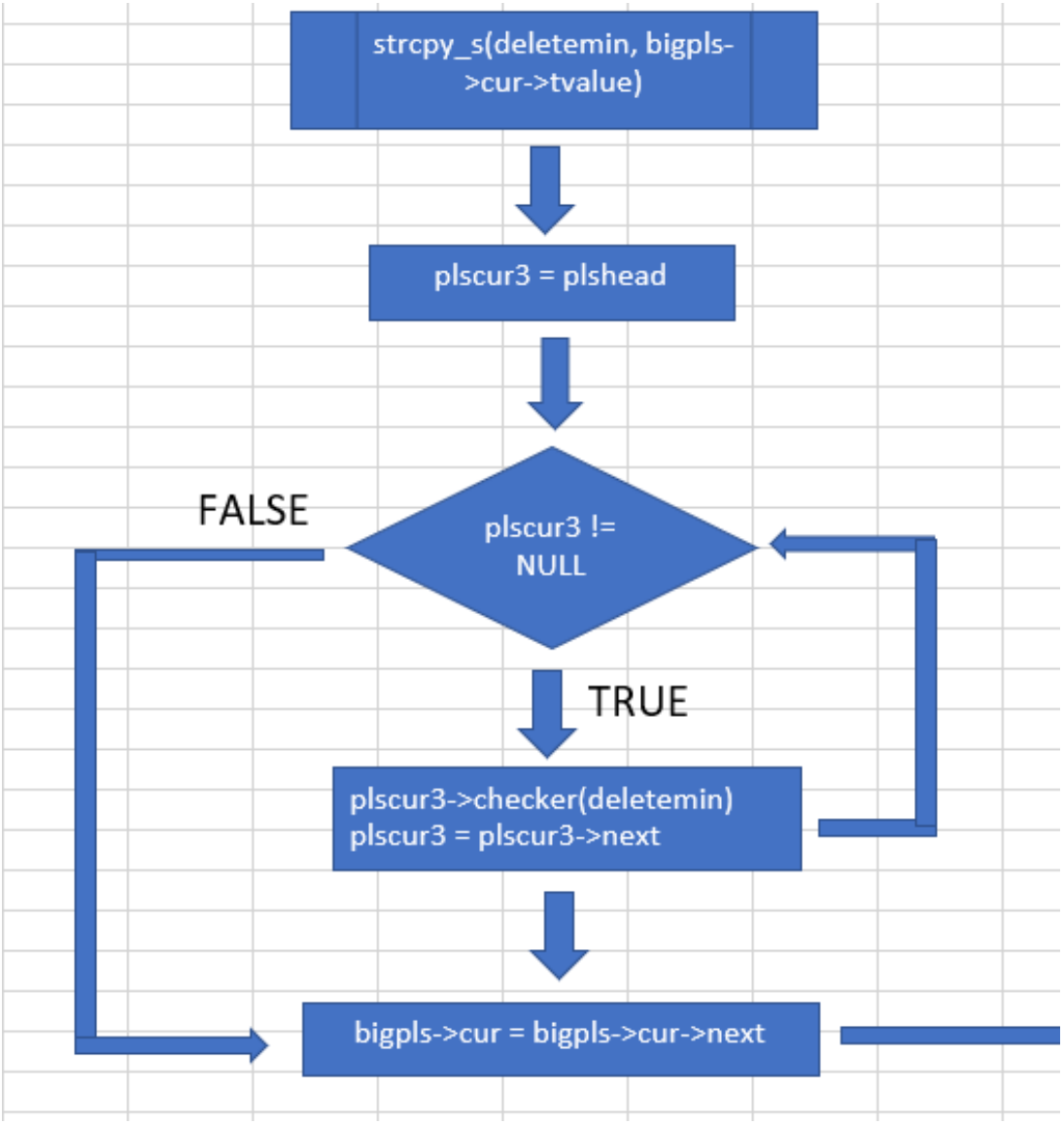


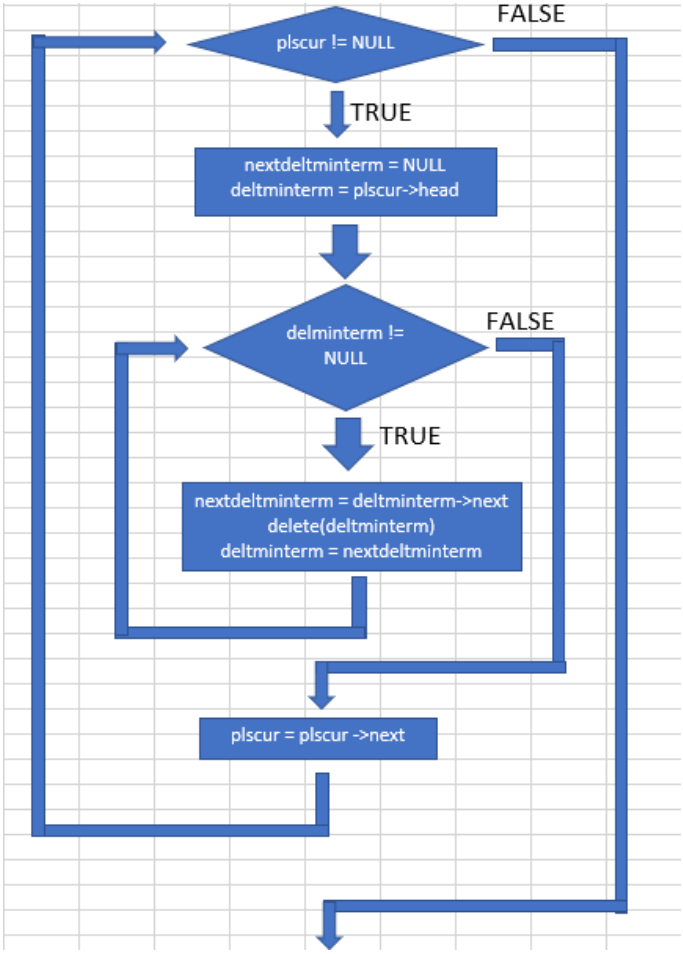
<트랜지스터 개수 계산>


essentialcur != NULL
TRUE
countandinput = 0
for i
0, 1, i<bitnum
essentialcur -> value[i]
== '0'
TRUE
checknot[i] = 1
FALSE
essentialcur -> value[i] == '0' ||
essentialcur -> value[i] == '1'
TRUE
countandinput ++
FALSE
for i
countand += countandinput * 2 + 2
countor++
essentialcur = essentialcur -> next
countor = countor * 2 + 2

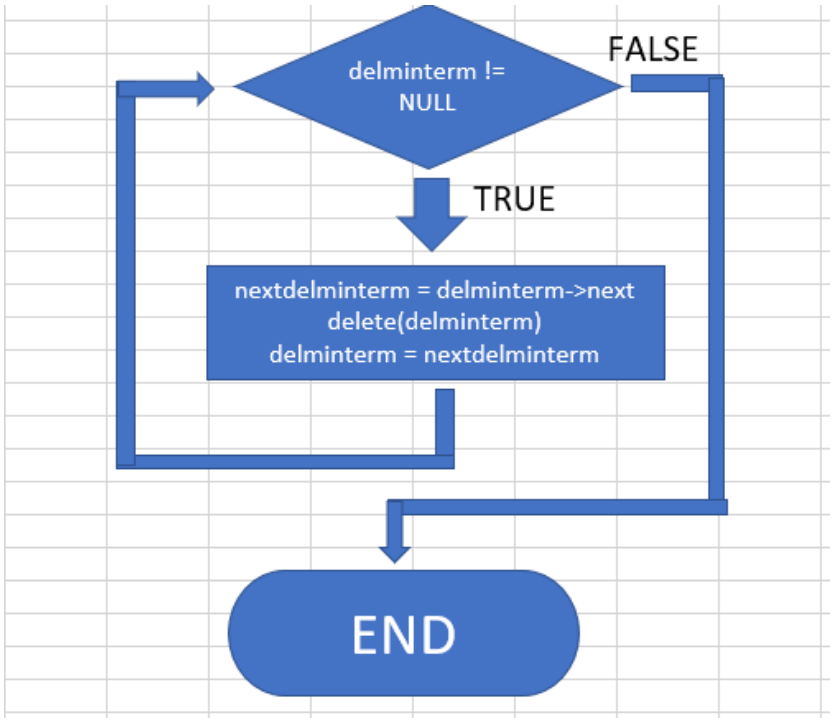

countor = countor * 2 + 2
for i
0, 1, i<bitnum
checknot[i] == 1
countnot++
for i
countnot = 2 * countnot
cost = countnot + countand + countor
Cost (# of transistors) :
cost출력
ofstream fout(result.txt)

<동적할당 해제>


strcpy_s(deletemin, bigpls->cur->tvalue)
plscur3 = plshead
plscur3 != NULL
FALSE
TRUE
plscur3->checker(deletemin)
plscur3 = plscur3->next
bigpls->cur = bigpls->cur->next


plscur != NULL
FALSE
TRUE
nextdeltminterm = NULL
deltminterm = plscur->head
delminterm != NULL
FALSE
TRUE
nextdeltminterm = deltminterm->next
delete(deltminterm)
deltminterm = nextdeltminterm
plscur = plscur ->next

<종료>

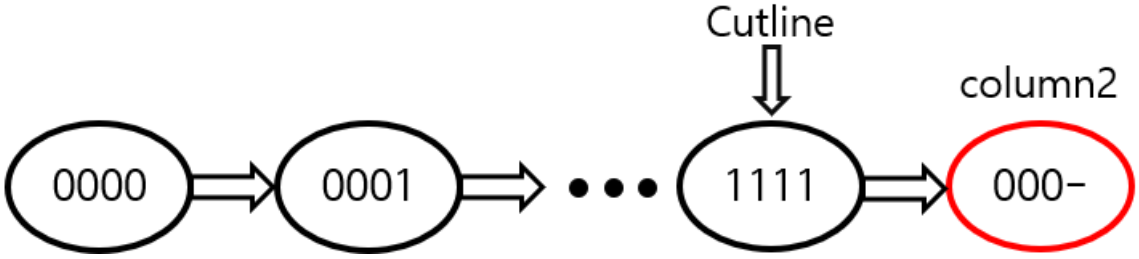

delminterm != NULL
FALSE
TRUE
nextdelminterm = delminterm->next
delete(delminterm)
delminterm = nextdelminterm
END

3) Verification strategy & corresponding examples with explanation

프로젝트 설명에 있었던 예시를 가져와 설명 드리겠습니다.

우선 input_minterm.txt파일을 열어 정보를 확인합니다. Linked list를 만들어 한 줄씩 getline을 통해 정보를 노드에 넣어줍니다. 이렇게 하면 가장 첫 노드의 value에는 4가 들어가게 되고 그 다음 노드의 value에는 d 0000이 들어가게 되고 이런 식으로 쭉 txt파일에 입력된 만큼 Linked list에 추가되게 됩니다. input정보가 전부 입력되고 나면 이후에 계산에 편의를 위해 d와 m을 value에서 삭제해줌과 동시에 따로 int dorm을 통해 이것이 True minterm인지 don't care인지 구분해줍니다.

이후에 계산을 하면서 다음 column으로 넘겨줘야 하는데 column마다 따로 list를 만들려고 했지만 input이 얼마나 들어올지 모르기 때문에 column의 수를 정할 수가 없었습니다. 그래서 저희가 생각해낸 방법은 cutline이라는 포인터를 통해 각 column들의 마지막 노드를 가리켜 column들을 구분하고 다음 column의 새로운 노드를 뒤에 계속 연결하는 방법을 구상하였습니다. 그림으로 설명하면 다음과 같습니다.

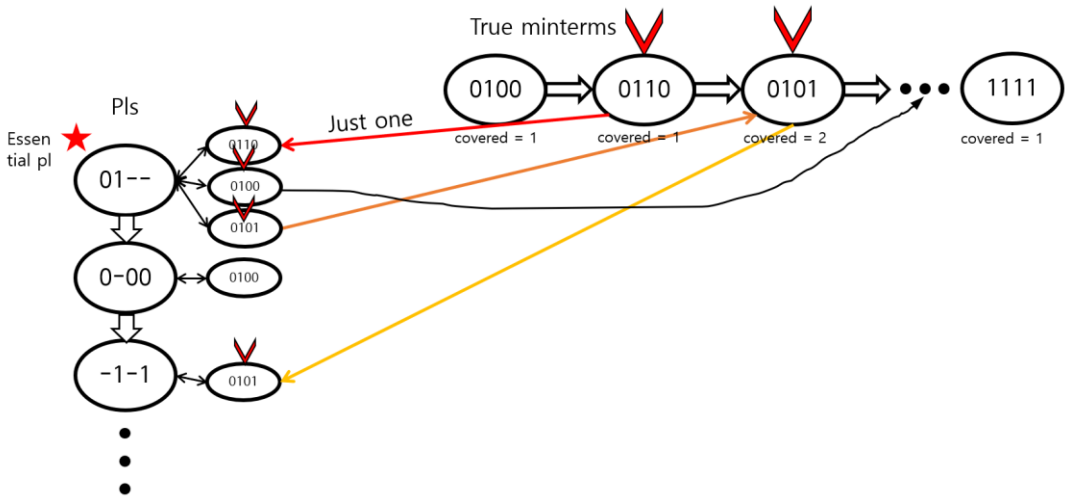


while (standard != NULL) 을 통해 첫 value부터 기준으로 잡고 쭉 끝까지 비교하게 됩니다. 새로운 노드가 생기면 계속 뒤로 이어지기 때문에 결국 모든 계산이 끝나서 더 이상 새로운 노드가 연결되지 않을 때 반복문이 종료되게 됩니다.

추가적으로 standard가 Cutline과 같아진 경우 한 column계산이 끝났다는 말이기 때문에 Cutline을 tail로 옮겨주어서 계속 나아갔습니다.

어떻게 계산하였는지에 대해 설명 드리겠습니다. standard(기준)을 잡고 나면 그 다음 노드부터 다시 while문을 돌립니다. 돌리면서 각자리를 비교하여 hamming distance가 1이라면 그 자리를 저장해 두었다가 그 자리에 '-'를 넣어주며 tail에 이어주고 check라는 변수를 1로 바꿔주어 조합에 사용됨을 표시해줍니다. 이렇게 하여 마지막 column까지 만들고 나면 check를 확인하여 조합되지 못한 value들을 pls로 연결해줍니다. 모든 노드를 1대1로 다 비교하였기 때문에 pls에는 중복된 노드가 생성 되어있을 수 있습니다. 그렇기 때문에 중복된 노드들을 삭제해 줍니다. pls를 정리해주고 나면 최종 계산을 위해 true minterm들을 tlist라는 linked list에 넣어주고 tlist와 pls를 비교하여 true minterm들은

자기가 몇 번 커버되는지 pls에 자기가 커버하는 minterm value들을 담아 줍니다.
그 후 essential pl을 찾기 위해 true minterm들의 커버된 수를 확인하여 1인 경우를 찾습니다. 찾은 다음 check변수를 1로 바꿔줍니다. 그리고 true minterm을 가지고 있는 pls를 찾아 그 pls가 커버하는 minterm들도 전부 check를 1로 바꿔줍니다. 그 다음에는 tlist에 있는 true minterm들이 전부 check가 1이 될 때까지 가장 많은 true minterm을 커버하는 pls를 찾아서 추가해주고 또 그 pls가 cover하는 true minterm들을 체크해줍니다. 그러면 최종 결과 값이 essentialplslist에 들어오게 됩니다. 그림으로 설명하면 다음과 같습니다. (빨주노 순으로 진행됩니다.)



최종 result가 01—, 1-01, 1010이라고 가정하면 이제 Cost를 계산하게 되는데 우선 not gate경우 각 자리별로 한번 만해주면 되기 때문에 char temp[100]{0,0, … ,0}을 만들어서 각 result들과 비교하여 0인 자리를 1로 바꿔준 다음 1의 개수를 세어 2N으로 계산하였고 or gate는 result들의 개수를 세어 2N+2로 계산해주었고 and gate는 각 result들의 '-'인 경우를 빼고 0과 1일때를 세어 2N+2로 계산해주었습니다. 0의 위치가 첫째자리 둘째자리 셋째자리 넷째자리에 다 있기 때문에 temp는 {1, 1, 1, 1}이 되어 not gate에서는 8개 A'B + A'C'D' + BD 각 수가 2개 3개 4개 이므로 and gate는 24개 그리고 총 3개의 합이므로 or gate는 8개입니다. 그 다음 다 더해주어서 result.txt파일에 최종 result들과 cost(40)를 저장해주고 동적할당 해제 후 종료됩니다.

4) TestBench

1) f (A,B,C,D,E,F,G) = ∑ m(64, 65, 69, 71, 74, 78)+ ∑ d(79)

(2021년도 1학기 중간고사 문제이며, bitnum이 크고 essential pI이 cover하는 다른minterm들을 처리하고 커버되지 못한 minterm을 위해 추가로 pls를 선택해야하는 케이스입니다.)

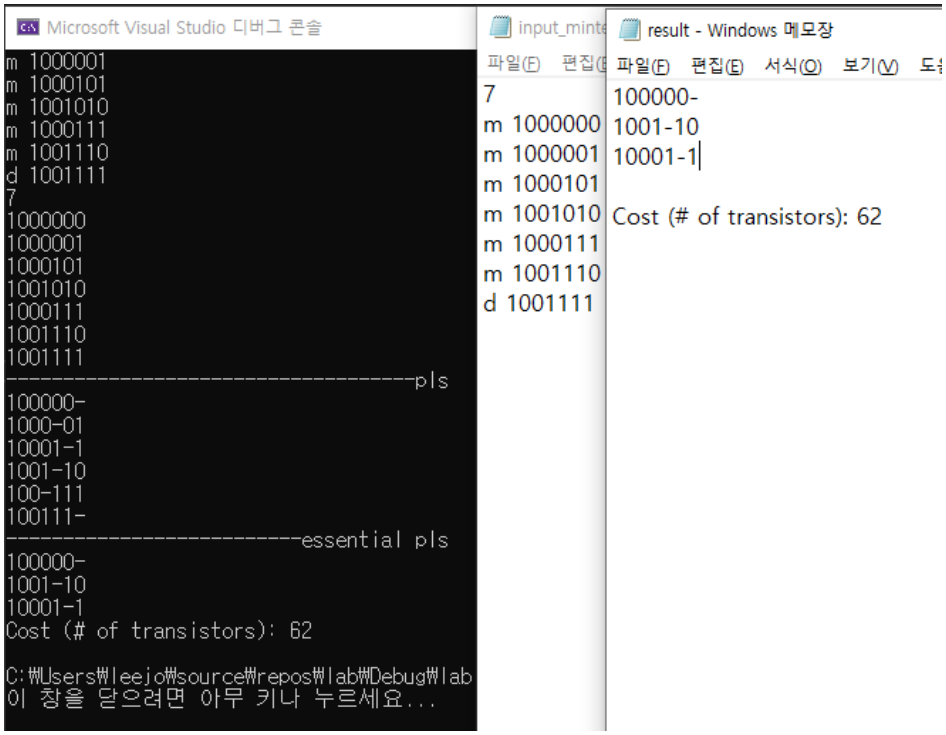
*: not cover

Column1	Column2	Column3
1000000	100000- *	
1000001	1000-01 *	
1000101	10001-1 *	
1001010	1001-10 *	
1000111	100-111 *	
1001110	100111- *	
1001111		

	1000000	1000001	1000101	1001010	1000111	1001110
100000-	x	x				
1000-01		x	x			
10001-1			x		x	
1001-10				x		x
100-111					x	
100111-						x

f = 100000-+10001-1+1001-10, f = AB'C'D'E'F'+AB'C'D'EG+AB'C'DFG'

6-input and gate(14)*3 + 3-input or gate(8) + not gate(2)*6 = 42+8+12 = 62



2) bitnum가 97인 크기가 큰 input이 들어왔을 경우