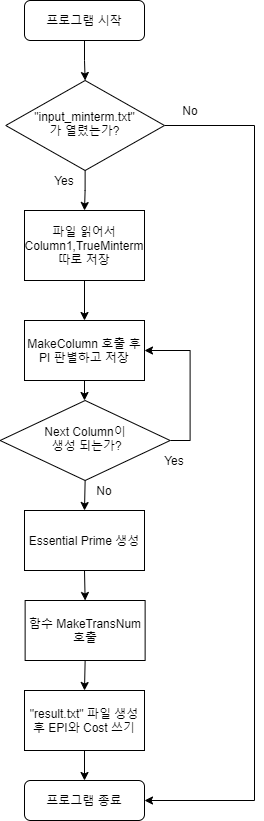
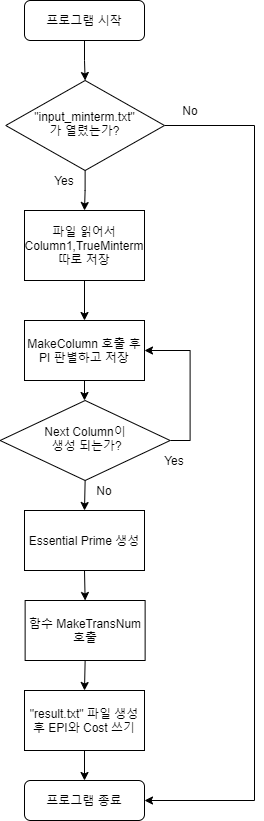
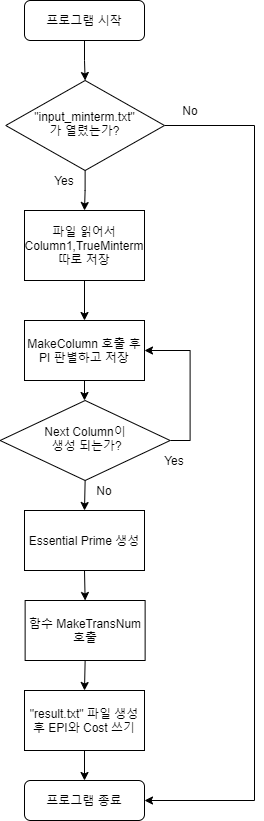
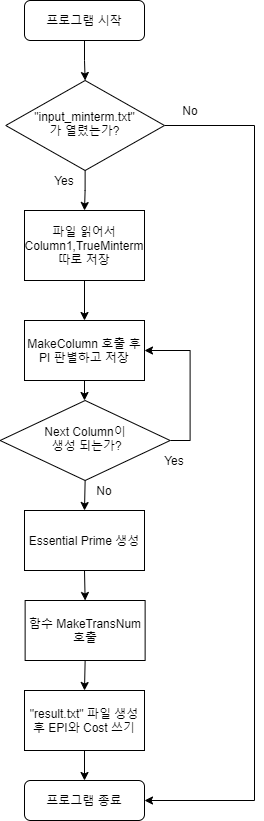
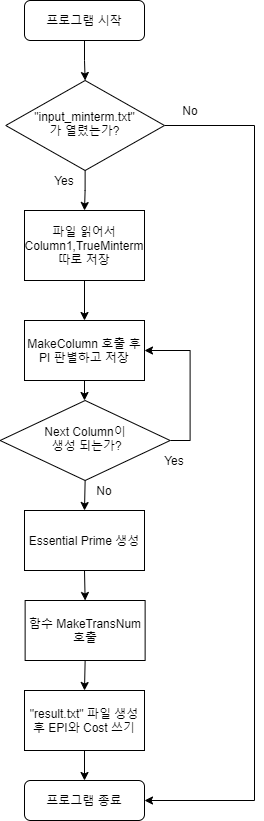
|  |
| --- |
| **Term Project**  **-Quine McCluskey-** |

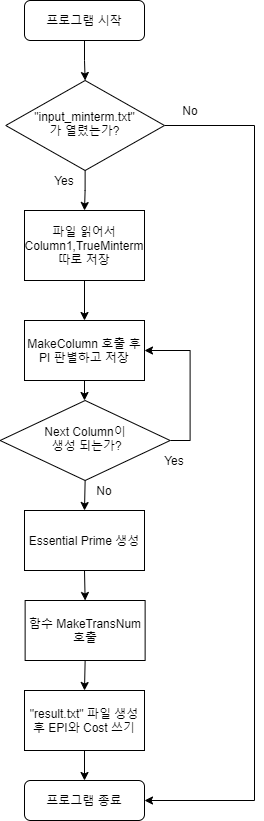
|  |  |
| --- | --- |
| **2021년 1학기** | |
| 제출일 | 2021.5.24 |
| 과목명 | 디지털 논리회로1 |
| 담당교수 | 이준환 |
|  | |
| 박지원 (2018202014) | |
| 정원섭 (2020202048) | |
| 나웅재 (2020202067) | |

* **Problem Statement**

input\_minterm.txt 파일로부터 데이터를 입력 받아 그것들의 Essential Prime Implicant들과 Cost of SOP를 출력해야 한다. input\_minterm 파일에 적혀있는 데이터들은 bit length와 value들이며, value들은 true minterm과 don’t care minterm으로 구분된다. Cost of SOP는 트랜지스터의 개수를 의미하며 inverter와 AND Gate, OR Gate에 사용된 트랜지스터의 개수를 합하여 계산한다. 출력은 result.txt 파일을 생성하여 결과 값들을 옮겨 적는다.

**Algorithm with flow chart****(동적 할당 및 메인 함수에 관한 pseudo code)**

**int main()**

**{**

**column1:칼럼 저장할 배열 선언;**

**column2:칼럼 저장할 배열 선언;**

**Prime\_Implicant:주항 저장할 배열 선언;**

**usage:칼럼 체킹할 배열 선언**

**true\_minterm:TrueMinterm 저장할 배열 선언;**

**파일 열기("input\_minterm.txt");**

**if (파일이 안열리면) {**

**"false" 출력;**

**프로그램 종료;**

**}**

**while (파일을 끝까지 읽을 때 까지)**

**{**

**한 줄 읽기;**

**if (true minterm인 경우) {**

**column1에 저장;**

**true\_minterm에 저장;**

**}**

**else if (Don't Care인 경우)**

**column1에 저장;**

**}**

**파일 닫기;**

**while (1)**

**{**

**if (짝수번 칼럼 일 경우)**

**{**

**column2에 칼럼 저장하는 함수(column2, column1, usage);**

**Prime\_Implicnat에 주항 판별 후 저장;**

**if (칼럼이 더 이상 생성이 안될 경우) break;**

**column1 초기화;**

**}**

**else if(홀수번 칼럼 일 경우)**

**{**

**column1에 칼럼 저장하는 함수(column1, column2, usage);**

**Prime\_Implicnat에 주항 판별 후 저장;**

**if (칼럼이 더 이상 생성이 안될 경우) break;**

**column2 초기화;**

**}**

**}**

**if (마지막 칼럼이 짝수 번 일 경우)**

**Prime\_Implicant에 column2 저장**

**else if (마지막 칼럼이 홀수 번 일 경우)**

**Prime\_Implicant에 column1 저장**

**EssentialPI : Essential Prime Implicant 저장할 배열 선언;**

**PITable:커버 확인행을 포함한 PITable 2차원 배열 선언;**

**for (PITable 마지막 행까지)**

**{**

**for (PITable 마지막 열까지)**

**PITable 0으로 초기화;**

**}**

**for (PITable 마지막 행까지)**

**{**

**for (PITable 마지막 열까지)**

**{**

**int count = 0;**

**for (주항 비트 수 만큼)**

**{**

**if (주항 각 비트 == 트루민텀 각 비트)**

**count++;**

**else if (주항 비트 == '-')**

**count++;**

**}**

**if (count == 주항 비트 수)**

**PITable에 1 마킹;**

**}**

**}**

**for (트루민텀 갯수 만큼)**

**{**

**int PIRowIdx, PIColumnIdx;**

**OneCount:각 열에 1의 갯수 카운팅 해주는 변수 선언**

**for (주항 갯수 만큼)**

**{**

**if (PITable에 1이 마킹된 경우)**

**OneCount++;**

**}**

**if (OneCount == 1 이고 PITable 확인행에 체크 되지 않은 경우)**

**{**

**EssentialPI에 해당 주항 저장;**

**for (트루민텀 갯수만큼)**

**{**

**if (PITable 해당 열에 1이 마킹 돼있는 경우)**

**{**

**for (주항 갯수 만큼)**

**해당 열 0으로 초기화;**

**해당 열 확인행 체크;**

**}**

**}**

**}**

**}**

**while (1)**

**{**

**if (확인행 전부 체크 됐는지 확인하는 함수(PITable, 주항 갯수, 트루민텀 갯수) == true)break;**

**MaxOne:주항이 가장 많은 트루민텀 체크하는 갯수 변수 선언**

**MaxIndex:MaxOne에 해당하는 열 인덱스 저장 변수 선언**

**Count\_1:주항이 트루민텀 체크하는 갯수 변수 선언**

**for (i<주항 갯수) //PIT확인행 체크된 열 초기화 후 남은 열 중 비교**

**{**

**Count\_1 = 0;**

**for (j<트루민텀 갯수)**

**{**

**if (PITable[i][j] == 1) //1의 갯수 체크**

**Count\_1++;**

**}**

**if (Count\_1 > MaxOne) //가장 많은 1의 갯수 체크**

**{**

**MaxOne = Count\_1;**

**MaxIndex = i;**

**}**

**}**

**for (i<트루민텀 갯수 만큼)**

**{**

**if (PITable[MaxIndex][i] == 1)**

**{**

**for (주항 갯수 만큼)**

**해당 열 0으로 초기화;**

**해당 열 확인행 체크;**

**}**

**}**

**EssentialPI에 해당 주항 저장;**

**}**

**writeResult : ofstream 개방**

**"result.txt" : 값을 출력할 파일 오픈**

**temp : 값들을 한꺼번에 출력하기 위한 string**

**for(EssentialPICount 배열에 저장된 모든 배열 요소에 대해 작용)**

**temp += EssentialPI 배열 요소 + '\n'**

**temp += "트랜지스터 개수 출력 안내 메세지"**

**result.txt << temp에 모아놨던 값**

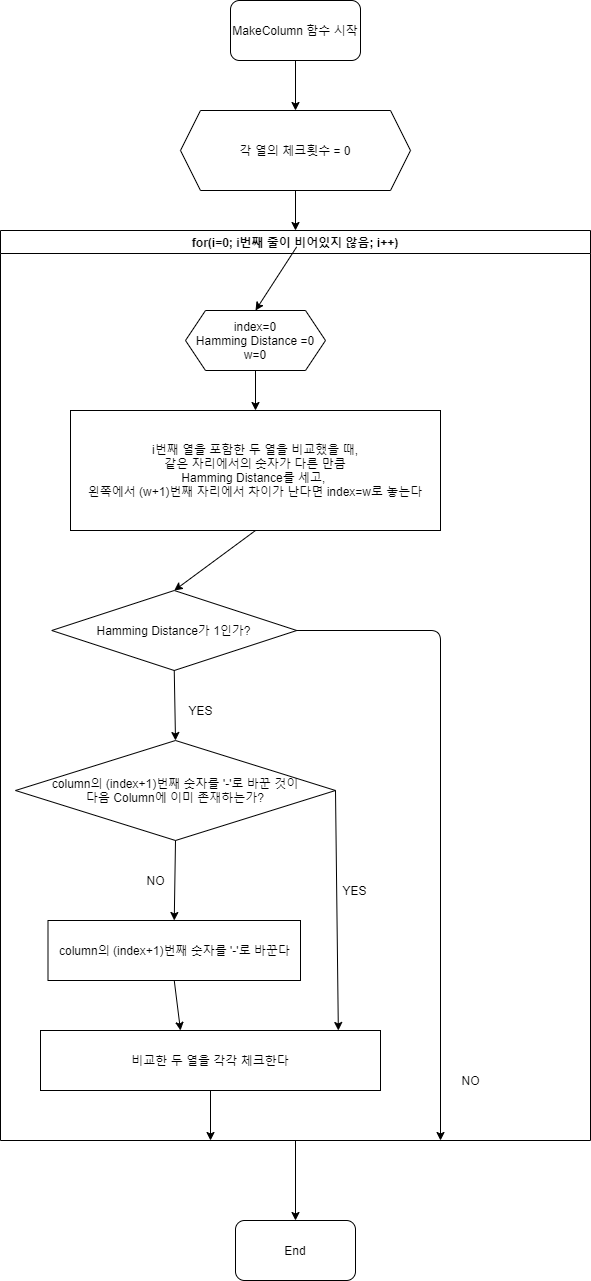
**result.txt << 트랜지스터 개수**

**파일 폐쇄**

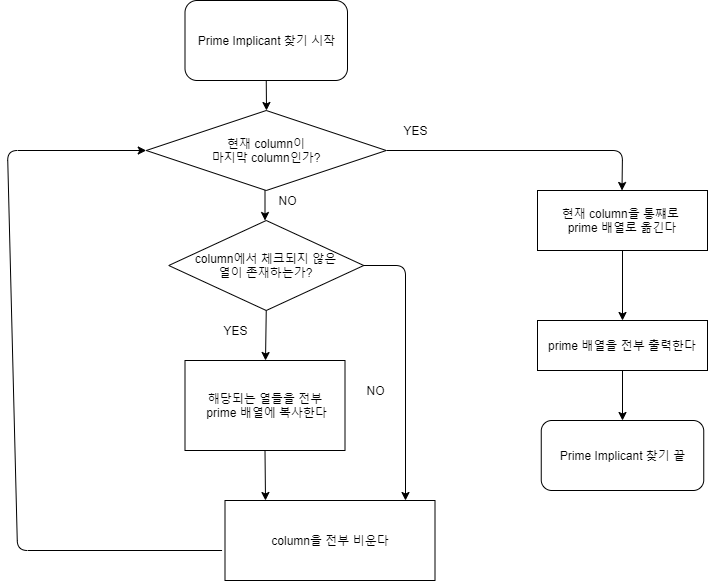
**지금까지 동적 할당 했던 메모리들 모두 해제**

**}**

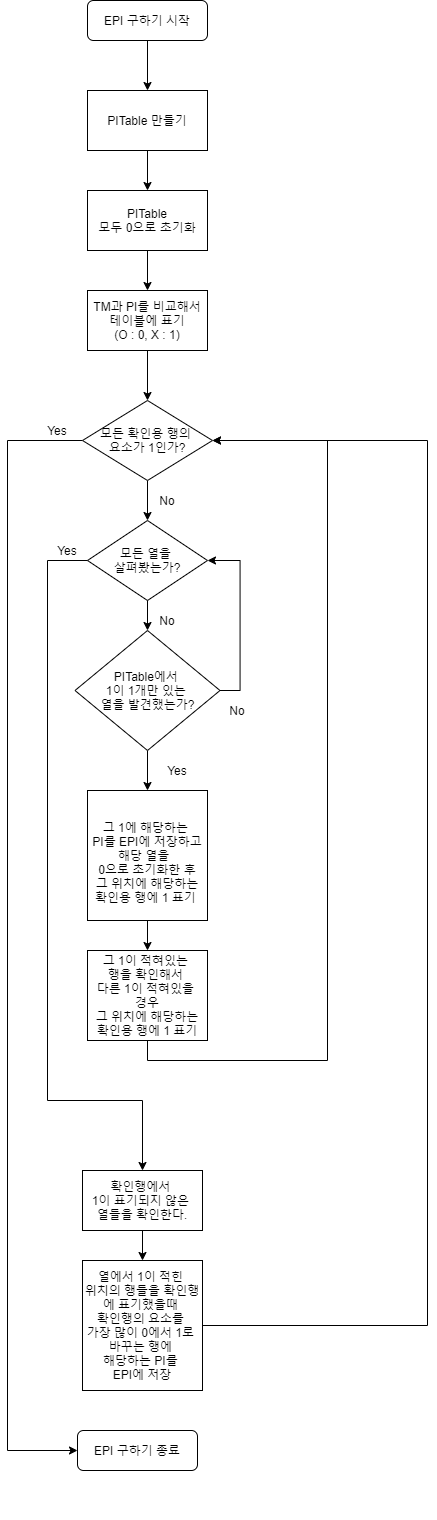
**(Prime Implicant 찾는 부분에 대한 pseudo code)**



**(Prime Implicant에 대한 pseudo code)**



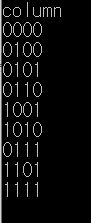
**(essential prime implicant 찾는 부분에 대한 pseudo code)**



* **Verification strategy & corresponding example with explanation**

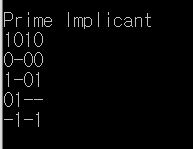
먼저 input\_minterm.txt로부터 column에 값들을 입력 받는다. 이때, m으로 구분되어 있는 값들은 true\_minterm 배열에 따로 저장한다. 나중에 PI테이블 만들 때 사용하기 위한 용도다.

< 현재 column 상태 >



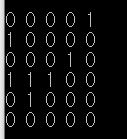
Column은 짝수번과 홀수번으로 구분한다. 그 후 PI을 구분하기 위해 반복을 시작한다. (Makecolumn을 호출해서 다음 열 만듦/ 수정 필요) 다음 Copyprime 함수를 사용하여 체크가 되어 있지 않은 배열 요소를 PI배열에 저장한다. 현재 Column에 더 이상 비교할 것이 없으면 마지막 Column임을 파악하고 배열 요소를 모두 PI배열에 옮긴 후 반복을 종료한다. 그렇지 않다면 이미 새로운 Column을 생성했으므로 기존에 사용하던 Column은 초기화한다.

< PI 구분 결과 >



그 후 처음에 분리해두었던 True Minterm들과 방금 구분한 PI를 이용하여 PITable을 만든다. 나중에 무한 반복문을 탈출할 조건에 사용하기 위한 확인행을 아래에 한 줄 더 추가한다.

<완성한 PITable>



EPI를 구현하기 위한 알고리즘은 이렇다.

1. 처음에 모든 열을 확인해서 1이 1개만 있는 열이 있는지 확인한다.

이 경우에는 3~5열이 해당한다.

2. 1이 한 개만 있는 열이 존재한다면 확인행에 1을 표기하고 1이 존재하는 행에 해당하는 PI를 EPI에 저장한다.

3. 만약 그 1이 존재하는 행을 확인해 다른 1이 있다면 그 위치에 해당하는 확인행에도 1을 표기한다.

4. 모든 확인행이 1이 되면 반복문을 탈출한다.

0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0

0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0

1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0

0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1

만약 위의 시행을 했는데도 확인행이 모두 1이 아니거나 1이 한 개만 있는 열이 없는 경우는 다음을 반복한다.

1. 행에서 1의 개수를 세어 가장 1의 개수가 많은 행에 해당하는 PI를 EPI에 저장하고 확인행에 표기한다. 만약 모두 같은 개수일 경우 아무거나 하나 정한다.

2. 확인행에서 0의 위치를 확인하고 그 열에 1이 존재하는 위치를 확인한다.

3. 그 1들이 존재하는 행을 확인하여 가장 많이 확인행을 1로 바꿀수 있는 PI를 EPI로 정한다.

1 1 0 0 0 1 1 0 0 0 1 1 0 0 0

0 1 1 0 0 0 1 1 0 0 (3열 1개) 0 1 1 0 0

0 0 1 1 0 0 0 1 1 0 (3열, 4열 2개 EPI) 0 0 1 1 0

0 0 0 1 1 0 0 0 1 1 0 0 0 1 1

1 0 0 0 1 1 0 0 0 1 1 0 0 0 1

1 1 0 0 0 1 1 1 1 0 1 1 1 1 1

^

<EPI 계산 완료>



트랜지스터 개수는 다음과 같이 계산한다.

인버터의 트랜지스터 개수는 0이 1번이라도 사용된 자리 개수 \* 2를 한다.

위 경우는 모든 자리에 0이 사용되었으므로 4 \* 2 = 8이다.

And게이트 트랜지스터 개수는 전체 0과 1의 개수 \* 2 + 2 \* EPI의 개수이다.

위 경우 0과 1은 (2 + 3 + 4)해서 총 9번 사용되었고 EPI의 개수는 3개이므로

9 \* 2 + 2 \* 3 = 24. 총 24개이다.

OR게이트 트랜지스터 개수는 EPI개수 \* 2 + 2 이므로

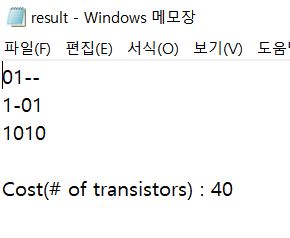
3 \* 2 + 2 = 8. 총 8개이다.

따라서 총 트랜지스터 개수는 위에 구한 값을 모두 합한 40개가 된다.

<트랜지스터 개수 계산 완료>



<프로그램 실행 결과>



* **A testbench that you think it is very hard to solve**

**<input\_minterm.txt>**

**5**

**m 00000**

**m 00010**

**m 00100**

**m 01000**

**m 10000**

**m 00110**

**m 01010**

**m 01100**

**m 10010**

**m 00111**

**m 01011**

**m 01101**

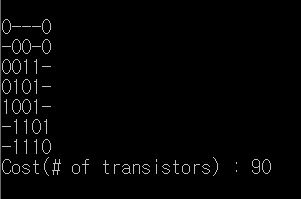
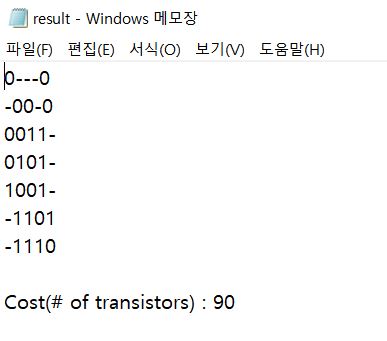
**m 01110**

**m 10011**

**m 11101**

**m 11110**

**<result.txt>**

****