

**UNIVERSIDADE DE BRASÍLIA**

**Wesley Ferreira Santos**

**(14/0033491)**

**David Guimarães Nakamura de**

**Lira (13/0007986)**

**Fernando Lemos de Carvalho**

**(13/0110361)**

**Fernando Campelo**

**(14/0020144)**

**Algoritmos e Estruturas de Dados – Turma B**

**TRABALHO FINAL**

Brasília, julho de 2016.

public class Quine\_McCluskey

{

public static void main (String args[])

{

//int[] minterms = {4, 5, 6, 8, 9, 10, 13};

//int[] dontCare = {0, 7, 15};

int[] minterms = {4,8,10,11,12,15};

int[] dontCare = {9,14};

int result[][] = quine(minterms, dontCare);

System.out.println("\nImplicantes primos essenciais: ");

print\_table(result);

}

/\*\*

\* REALIZA TODA A OPERAÇÃO NECESSáRIA PARA O QUINE MCCLUSKEY, ENCONTRANDO OS IMPLICANTES PRIMOS

\* E OS IMPLICANTES PRIMOS ESSENCIAIS

\*

\* @param MINTERMS MINETERMOS INICIAIS

\* @param DONTCARE DONTCARE'S INICIAIS

\* @return MATRIZ DE IMPLICANTES PRIMOS ESSENCIAIS

\*/

public static int[][] quine(int minterms[], int dontCare[])

{

int allTerms[] = new int[minterms.length + dontCare.length];

for(int i = 0; i < minterms.length; i++)

{

allTerms[i] = minterms[i];

}

for(int i = 0; i < dontCare.length; i++)

{

allTerms[allTerms.length - i - 1] = dontCare[i];

}

int tabela[][] = create\_table(allTerms);

int tabelaMinterms[][] = create\_table(minterms);

int tabelaAux1[][] = create\_reduced\_matrix(tabela);

int tabelaAux2[][] = create\_reduced\_matrix(tabelaAux1);

int notUsedMinterms1[][] = not\_used\_minterms(tabelaAux1);

int notUsedMinterms2[][] = not\_used\_minterms(tabelaAux2);

System.out.println("\nColuna 1");

print\_table(tabela);

System.out.println("\nColuna 2");

print\_table(tabelaAux1);

// System.out.println("\nMini termos não usados coluna 2");

// print\_table(notUsedMinterms1);

System.out.println("\nColuna 3");

print\_table(tabelaAux2);

//

// System.out.println("\nMini termos não usados coluna 3");

// print\_table(notUsedMinterms2);

int [][] prime\_implicant\_table = result\_implicant\_table(notUsedMinterms1, notUsedMinterms2);

return prime\_implicant\_chart(prime\_implicant\_table, tabelaMinterms);

}

/\*\*

\* DADAS UMA MATRIZ DE IMPLICANTES PRIMOS E UMA OUTRA DE MINITERMOS INICIAIS, EXCLUINDO OS "DON'T CARE",

\* RETORNA OS IMPLICANTES PRIMOS ESSENCIAIS

\*

\* @param not\_used\_minterms MATRIZ DE MINETERMOS QUE AINDA NÃO FORAM USADOS

\* @param initial\_minterms MATRIZ DE MINETERMOS DE ENTRADA DO PROGRAMA

\* @return MATRIZ DE IMPLICANTES PRIMOS ESSENCIAIS

\*/

//TODO: DESCOBRIR OS IMPLICANTES PRIMOS QUE FALTAM PARA QUE TODOS OS MINITERMOS SEJAM COBERTOS

public static int[][] prime\_implicant\_chart(int not\_used\_minterms[][], int[][] initial\_minterms)

{

// system.out.println("not used minterms");

// print\_table(not\_used\_minterms);

// system.out.println("tabela minterms");

// print\_table(initial\_minterms);

int newMatrix[][] = new int [initial\_minterms.length][not\_used\_minterms.length];

//PERCORRE A MATRIZ E COLOCA TUDO COMO -1

for(int i = 0; i < newMatrix.length; i++)

{

for(int j = 0; j < newMatrix[i].length; j++)

{

newMatrix[i][j] = -1;

}

}

//COMPARA TODOS OS MINITERMOS INICIAIS COM OS MINITERMOS NÃO USADOS

int prime\_implicant\_added = 0; //MUDAR O INDEX DOS ELEMENTOS QUE SERÃO ADICIONADOS NA MATRIZ "newmatrix"

for(int i = 0; i < initial\_minterms.length; i++) //PERCORRE TODOS OS MINITERMOS INICIAIS

{

for(int j = 0; j < not\_used\_minterms.length; j++) //PERCORRE TODOS OS MINITERMOS NÃO USADOS

{

for(int k = 0; k < 4; k++) //PERCORRE E COMPARA BIT A BIT OS ELEMENTOS DAS MATRIZES initial\_minterms E not\_used\_minterms

{

//SE K FOR IGUAL A 3 (ULTIMO BIT) E not\_used\_minterms[J][K] == 2 (not\_used\_minterms[J][K] == '-')

// OU initial\_mintermS[I][K] == not\_used\_minterms[J][K] (O BIT EM initial\_minterms[I][K] É IGUAL AO BIT EM not\_used\_minterms[I][K])

//ENTAO ADICIONA O INDICE DO not\_used\_minterms NA MATRIZ "newmatrix"

//K == 3 SIGNIFICA QUE TODOS OS BIT ANTERIORES Já FORAM CONFERIDOS E PERMITEM QUE O MINITERMO

//PODE REPRESENTAR O MINETERMO initial\_minterms[I]

if(k == 3 && ((not\_used\_minterms[j][k] == 2) || (initial\_minterms[i][k] == not\_used\_minterms[j][k])))

{

//ADICIONA INDICE DO not\_used\_minterms QUE FOI VERIFICADO E PODE REPRESENTAR initial\_minterms[I]

newMatrix[i][prime\_implicant\_added] = j;

//INCREMENTA prime\_implicant\_added PARA QUE NA PROXIMA INSERÇÃO NA MATRIZ NÃO OCORRA EM CIMA DO INDICE QUE FOI GRAVADO AGORA

prime\_implicant\_added++;

//PARA O LAÇO DE REPETIÇÃO DE "K"

break;

}

//SE not\_used\_minterms[J][K] == 2 (not\_used\_minterms[J][K] == '-'), NÃO IMPORTA O VALOR DE initial\_minterms[I], POIS SERá VALIDO TANTO PARA '0'

//QUANDO PARA '1', ENTAO USA-SE O 'CONTINUE' PARA PROSSEGUIR NO LAÇO DE 'K'

if(not\_used\_minterms[j][k] == 2) continue;

//SE initial\_minterms[I][K] == not\_used\_minterms[J][K] (0 == 0 OU 1 == 1), ENTAO USA-SE O 'CONTINUE' PARA PROSSEGUIR NO LAÇO DE 'K'

if(initial\_minterms[i][k] == not\_used\_minterms[j][k]) continue;

//SE NENHUMA DESSAS OPÇÕES FOREM SATISFEITAS PARA O LAÇO DE REPETIÇÃO DE 'K', É PROVAVEL QUE NUNCA CHEGUE AQUI,

//MAS É BOM USAR POR UMA QUESTAO DE SEGURANÇA

break;

}

}

//QUANDO TERMINAR DE COMPARAR initial\_minterms[I] COM TODOS OS not\_used\_minterms ENTAO prime\_implicant\_added = 0

//ISSO SERVE PARA QUE OS INDICES QUE FOREM ADICIONADOS NA newmatrix COMEÇEM A SER ADICIONADOS NO INDICE 0

prime\_implicant\_added = 0;

}

//ARRAY QUE GUARDA O COMPRIMENTO DE CADA ARRAY NO ARRAY DE ARRAY'S "newmatrix"

//SERVIRá PARA SABER QUAL O MINITERMO PODE SER REPRESENTADO APENAS POR UM IMPLICANTE PRIMáRIO

int usedIndex[] = new int[newMatrix.length];

for(int i = 0; i < newMatrix.length; i++)

{

int length = 0;

for(int j = 0; j < newMatrix[i].length; j++)

{

//OS ELEMENTOS QUE GUARDAM -1 NÃO DEVERÃO SER CONTADOS E REPRESENTAM O FIM DO ARRAY

if(newMatrix[i][j] != -1) length++;

else break;

}

usedIndex[i] = length;

}

System.out.println("\nImprime X's");

for(int i = 0; i < newMatrix.length; i++)

{

for(int j = 0; j < usedIndex[i]; j++)

{

System.out.print(newMatrix[i][j] + " ");

}

System.out.println();

}

//SE TIVER APENAS UM 1 IMPLICANTE PRIMO, PASSAR A SER UM IMPLICANTE PRIMO ESSENCIAL

//OU SEJA, SE usedindex[I] == 1

int essential\_implicant[][] = new int[newMatrix.length][];

System.out.println();

//CONTA QUANTOS IMPLICANTES PRIMOS ESSENCIAIS FORAM CRIADOS

int essential\_added = 0;

for(int i = 0; i < usedIndex.length; i++)

{

System.out.println(usedIndex[i]);

if(usedIndex[i] == 1)

{

essential\_implicant[essential\_added] = not\_used\_minterms[newMatrix[i][0]];

essential\_added++;

}

}

// int newMatrixWithoutEssential[][] = new int [usedIndex.length - essential\_added][not\_used\_minterms.length];

return create\_matrix\_from\_biggest\_matrix(essential\_implicant, essential\_added);

}

/\*

RECEBE DUAS MATRIZES E ADICIONA AS LINHAS DA matrix1 E DEPOIS AS DA matrix2 EM UMA NOVA MATRIZ

CONFERINDO ANTES SE A LINHA JA NAO EXISTE NA newMatrix PARA QUE NAO HAJA LINHAS REPETIDAS

@param matrix1 primeira matriz

@param matrix2 primeira matriz

@return matrix1 + matrix2, SEM REPETIR ELEMENTOS

\*/

public static int[][] result\_implicant\_table(int matrix1[][], int matrix2[][])

{

int newMatrix[][] = new int[50][];

int itensAdded = 0;

for(int i = 0; i < matrix1.length; i++)

{

//SE ITEM NAO EXISTE NA MATRIZ, ENTÃO ADICIONA NA MATRIZ

if(!item\_exist\_in\_matrix(newMatrix, matrix1[i]))

{

newMatrix[itensAdded] = matrix1[i];

itensAdded++;

}

}// NESSE PONTO, TODAS AS LINHAS DA matrix1 COMPOEM A newMatriz QUE NAO DEVE LINHAS REPETIDAS

//ESTE LOOP ADICIONA AS LINHAS DA matrix2 NA newMatrix A PARTIR

//DA POSICAO itensAdded QUE CRESCEU NO LOOP ANTERIOR(ADICIONANDO AS LINHAS DA matrix1)

for(int i = 0; i < matrix2.length; i++)

{

if(!item\_exist\_in\_matrix(newMatrix, matrix2[i]))

{

newMatrix[itensAdded] = matrix2[i];

itensAdded++;

}

}

return create\_matrix\_from\_biggest\_matrix(newMatrix, itensAdded);

}

//FUNÇÃO BOOLEANA QUE RECEBE UMA MATRIZ E UM ARRAY RETORNA VERDADEIRO SE EXISTE ALGUMA LINHA DA MATRIZ IGUAL AO ARRAY.

//FUNÇÃO GENÉRICA.

public static boolean item\_exist\_in\_matrix(int matrix[][], int term[])

{

for (int i = 0; i < matrix.length; i++)

{

if(matrix[i] == null) break;

if(matrix[i][0] == term[0] && matrix[i][1] == term[1] && matrix[i][2] == term[2] && matrix[i][3] == term[3])

{

return true;

}

}

return false;

}

//COMPARA CADA LINHA DE UMA MATRIZ COM TODAS AS OUTRAS A FIM DE CRIAR UMA NOVA MATRIZ EM QUE SUAS LINHAS SAO MINTERMOS

//QUE NAO TIVERAM APENAS UMA DIFERENCA COM QUALQUER OUTRO (NO METODO TABULAR FEITO COM PAPEL E CANETA,

//O PROCEDIMENTO DE MARCAR UMA LINHA EH EQUIVALENTE A ESTA FUNCAO).

public static int[][] not\_used\_minterms(int matrix[][])

{

int newMatrix[][] = new int[50][];

int minTermsNotUsed = 0;

for(int i = 0; i < matrix.length - 1; i++)

{

boolean minTermUsed = false;

for(int j = 0; j < matrix.length; j++)

{

if(qnt\_of\_different\_terms(matrix[i], matrix[j]) == 1)

{

minTermUsed = true;

break;

}

}

if(minTermUsed) continue;

newMatrix[minTermsNotUsed] = matrix[i];

minTermsNotUsed++; //QUANTIDADE DE LINHAS NAO USADAS QUE SERA O NUMERO DE LINHAS DA MATRIZ DA SAIDA

}

return create\_matrix\_from\_biggest\_matrix(newMatrix, minTermsNotUsed);

//EXEMPLO: SE ESSA FUNCAO RECEBESSE UMA MATRIZ A[][]={{0,0,0,0}, {0,0,0,1}, {0,0,1,0}, {1,1,1,1}},

//O RETORNO SERIA UMA NOVA MATRIZ B CONTENDO APENAS A LINHA {1,1,1,1} DA MATRIZ PARAMETRO.

}

//ORDEM DE USO 6 => RECEBE UMA MATRIZ E O NUMERO DE LINHAS DA MATRIZ QUE SERá CRIADA. FUNÇÃO GENÉRICA.

public static int[][] create\_matrix\_from\_biggest\_matrix(int matrix[][], int realUsedLines)

{

int newMatrix[][] = new int[realUsedLines][];

for(int i = 0; i < realUsedLines; i++)

{

newMatrix[i] = matrix[i];

}

return newMatrix; //RETONA UMA MATRIZ

}

//ORDEM DE USO 7 => FUNCAO QUE TORNA DESNECESSARIA A SEPARACAO DOS MINTERMOS EM GRUPOS. COMPARA TODAS AS LINHAS DA

// MATRIZ INICIAL E RETORNA UMA NOVA MATRIZ EM QUE SUAS LINHAS SÃO ARRAYS

//EM QUE O COMPUTADOR ENCONTROU APENAS UMA DIFERENCA NAS QUATRO COLUNAS. NESTE ALGORITMO NAO É NECESSARIO SEPARAR OS

//MINTERMOS EM GRUPOS PORQUE ESSA DIVISÃO SÓ SERVE PARA FACILITAR A VISUALIZAÇÃO DO DIGITOS NO PAPEL E APROXIMAR

//AS LINHAS QUE PROVAVELMENTE TERÃO UMA SO UM ELEMENTO DE DIFERENCA.

public static int[][] create\_reduced\_matrix(int matrix[][])

{

int newMatrix[][] = new int[50][];

int mintermsCreated = 0;

for(int i = 0; i < matrix.length - 1; i++)

{

for(int j = i + 1; j < matrix.length; j++)

{

if(qnt\_of\_different\_terms(matrix[i], matrix[j]) == 1)

{

newMatrix[mintermsCreated] = create\_minterm\_from\_terms(matrix[i], matrix[j]);

mintermsCreated++; //QUANTIDADE DE LINHAS DA PROXIMA MATRIZ CRESCENDO. SERVIRA DE PARAMENTRO

//PARA CRIACAO DA PROXIMA MATRIX.

}

}

}

return create\_matrix\_from\_biggest\_matrix(newMatrix, mintermsCreated); //EXEMPLO: SE O PARAMETRO FOSSE A MATRIZ

// matrix[][] = {{0,0,0,0}, {0,0,0,1}, {1,1,1,1}}, O RETORNO SERIA UMA MATRIZ F[][]={{0,0,0,2}},

// QUE SE IMPRESSA, MOSTRARIA 000-

}

//ORDEM DE USO 5 => RECEBE E COMPARA OS DIGITOS DA MESMA COLUNA DE DOIS ARRAYS (QUE INICIALMENTE SÃO AS LINHAS DA MATRIZ INICIAL)

//E RETORNA UM ARRAY COM O DIGITO QUE SE DEFERENCIOU SUBSTITUIDO POR 2.

public static int[] create\_minterm\_from\_terms(int minterms1[], int minterms2[])

{

int newTerm[] = new int[minterms1.length];

for(int i = 0; i < minterms1.length; i++)

{

if(minterms1[i] == minterms2[i])

{

newTerm[i] = minterms1[i];

}

else

{

newTerm[i] = 2;

}

}

return newTerm; //EXEMPLO SE ESTA FUNCÃO RECEBESSE A[] = {0,0,0,0} E B[] = {0,1,0,0}, ELA RETORNARIA UM ARRAY C[] = {0,2,0,0}.

}

//IMPRIME UMA MATRIZ DE FORMA QUE OS ELEMENTOS IGUAIS A 2 SEJAM IMPRESSOS COMO TRAÇO "-"

public static void print\_table(int table[][])

{

for(int j = 0; j < table.length; j++)

{

print\_minterm(table[j]);

System.out.println();

}

}

//FUNÇÃO USADA MAIS DE UMA VEZ. ELA RECEBE UM ARRAY E SE ALGUM DOS ELEMENTOS FOR O NÚMERO 2

//É IMPRESSO (APENAS IMPRIME, NAO SUSTITUE) "-" NO LUGAR DELE.

public static void print\_minterm(int minterm[])

{

for(int i = 0; i < minterm.length; i++)

{

if(minterm[i] == 2)

System.out.print("-"); //SE RECEBESSE a[] = {0,2,1,2}, SERIA IMPRESSO 0-1-

else System.out.print(minterm[i]);

}

}

//RECEBE DOIS ARRAYS (QUE SAO DUAS LINHAS DE UMA MATRIZ) COMO PARAMETRO E RETORNA O NUMERO DE TERMOS QUE SAO DIFERENTES

public static int qnt\_of\_different\_terms(int minterms1[], int minterms2[])

{

int qnt\_of\_different\_terms = 0;

for(int i = 0; i < 4 ; i++)

{

if(minterms1[i] != minterms2[i])

{

qnt\_of\_different\_terms++;

}

}

return qnt\_of\_different\_terms;// SE OS PARAMETROS FOSSEM A[] = {0, 0, 0, 1} e B[] = {1, 1, 1, 1}, A FUNÇÃO RETORNARIA 3,

//CASO FOSSEM C[] = {0, 1, 0, 0} e D[] = {0, 1, 0, 1}, A FUNÇÃO RETORNARIA 1.

}

//ORDEM DE USO 1 => RECEBE UM NUMERO INTEIRO/MINTERMO E O CONVERTE EM UM BINARIO DE 4 DIGITOS EM FORMA DE STRING

public static String decimal\_to\_binary\_string\_4\_digits(int decimal\_minterm)

{

String strBinaryNumber = Integer.toBinaryString(decimal\_minterm);

while (strBinaryNumber.length() < 4)

{

strBinaryNumber = "0" + strBinaryNumber;

}

return strBinaryNumber; // EXEMPLO: SE A FUNCAO RECEBEU 4, ELA RETORNARA UMA STRING "0100".

}

//ORDEM DE USO 2 => RECEBE UMA STRING CONTENDO OS DIGITOS EM BINARIO (EX:0001 OU 1111) E RETORNA UM ARRAY COM ESTES DIGITOS.

public static String[] binary\_digits\_to\_array(String strBinaryNumber)

{

String binary\_digits\_array[] = strBinaryNumber.split("");

return binary\_digits\_array; // RETORNA ARRAY DE STRING EX: array[] = {0, 0, 0, 1} ou array[] = {1, 1, 1, 1};

}

//ORDEM DE USO 3 => RECEBE UM ARRAY DE STRING CONTENDO OS DIGITOS BINARIOS DE UM NUMERO E RETORNA ESTE ARRAY NO TIPO INTEIRO.

public static int[] string\_to\_int\_array(String[] binary\_digits\_array)

{

int int\_binary\_array[]= new int[binary\_digits\_array.length];

for(int i = 0; i < binary\_digits\_array.length; i++)

{

int\_binary\_array[i] = Integer.parseInt(binary\_digits\_array[i]);

}

return int\_binary\_array; // EXEMPLO: RECEBEU string array[] = { 0, 1, 0, 1} e retorna int array[] = { 0, 1, 0, 1}.

}

//ORDEM DE USO 4 => RECEBE TODOS OS ARRAYS REFERENTES AOS MINTERMOS (EX: int array[] = { 0, 0, 1, 1}) E COM UM LAÇO OS TRANSFORMA

// NAS LINHAS DE UMA MATRIZ DE INTEIROS

public static int[][] create\_table(int[] minterms)

{

int[][] tabela = new int[minterms.length][3];

for(int i = 0; i < minterms.length; i++)

{

tabela[i] = string\_to\_int\_array(binary\_digits\_to\_array(decimal\_to\_binary\_string\_4\_digits(minterms[i])));

}

return tabela; //RETORNA MATRIZ EX: int matriz[][] = {{0, 0, 0, 0},

// {0, 0, 0, 1}...

// {1, 1, 1, 1}}

}

}