# ASSIGNMENT / PROJECT

## COURSE: DISCRETE MATHEMATICS (CS-211)

## GROUP MEMBERS:

* Munir Hassan (K16-3761)
* Basit Ali (K16-3762)
* Malik Obaid Ur Rehman (K16-3767)

## TOPICS:

* A program that generates a truth table of any expression having logical connectives and shows whether it is a tautology, contingency or a contradiction
* A program that generates a power set of a set

## LANGUAGE USED:

* JAVA

## CODE:

• A program that generates a truth table of any expression having logical connectives and shows whether it is a tautology, contingency or a contradiction

package truthtableprint;

import java.util.\*;

public class TruthTablePrint {

static int rows,columns;

static char ascii='a';

static int stepNo=1;

static int[][] TwoDArray;

public static void main(String[] args) {

Scanner scanf=new Scanner(System.in);

System.out.println("\n\n----------------INSTRUCTIONS----------------\n\n1.Program can handle maximum of 26 variables Expression(English Alphabets Allowed).\n"+

"2.The Porgram will accept All english alphabets, every type of Paranthesis and all logical Connectives Mentioned below :- \n\n"+

" ^ means AND/CONJUNCTION\n" +

" | means OR/DISJUNTCION\n" +

" @ means Implication\n" +

" - means Negation\n"+

" ~ means Biconditional\n"+

" + means XOR\n\n"+

"3.No other logical connectives and special variables are allowed in the program\n\n\n");

System.out.println("Enter a Expression : ");

String s=scanf.nextLine(),expression;

expression= s.replaceAll("\\s+", "");

expression=expression.toUpperCase();

if(paranthesisMatching(expression) && isAdjacentCharacter(expression)){

StringBuilder temp=new StringBuilder(expression);

expression=adjustParanthesis(temp);

String convertedExpression=expressionToEvaluate(expression);

if(!"\*".equals(convertedExpression)){

char[] variables=returnVariableArray(convertedExpression);

Arrays.sort(variables);

int count=returnCountOfVariables(variables);

int[][] array=printTruthTable(count,variables);

evaluatingTheExpression(convertedExpression,array,variables);

}

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

}

}

public static char[] returnVariableArray(String s){

int k=0;

char[] variables=new char[100];

Arrays.fill(variables, '`');

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

char ch=s.charAt(i);

if(isVariable(ch)){

if(!presentInArray(variables,ch)){

variables[k++]=ch;

}

}

}

return variables;

}

public static int returnCountOfVariables(char arr[]){

int count=0;

for(int i=0;arr[i]!='`';i++){

count++;

}

return count;

}

public static boolean presentInArray(char arr[],char ch){

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

if(arr[i]==ch){

return true;

}

}

return false;

}

public static String expressionToEvaluate(String s){

String finalExpression="";

char ch,te='0';int tag=0;

Stack<Character> s1=new Stack<>();

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

ch=s.charAt(i);

if(isVariable(ch)){

finalExpression+=ch;

}

else{

if(isOpeningParanthesis(ch)){

s1.push(ch);

te =ch;

}

else{

if(isClosingParanthesis(ch)){

if(ch==']'){

while(s1.peek()!='[' && !s1.isEmpty()){

finalExpression+=s1.pop();

}

s1.pop();

}

else{

if(ch=='}'){

while(s1.peek()!='{' && !s1.isEmpty()){

finalExpression+=s1.pop();

}

s1.pop();

}

else{

if(ch==')'){

while(s1.peek()!='(' && !s1.isEmpty()){

finalExpression+=s1.pop();

}

s1.pop();

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

finalExpression="\*";

tag=1;

break;

}

}

}

}

else{

if(isLogicalOperator(ch)){

while(!s1.isEmpty() && s1.peek()!=te && hasHigherPrecedence(ch,s1.peek())){

finalExpression+=s1.pop();

}

s1.push(ch);

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

finalExpression="\*";

tag=1;

break;

}

}

}

}

}

if(tag==0){

while(!s1.isEmpty()){

finalExpression+=s1.pop();

}

}

return finalExpression;

}

public static boolean paranthesisMatching(String exp){

int i,tag=0;

char ch;

Stack <Character>st=new Stack<>();

for(i=0;i<exp.length();i++){

ch=exp.charAt(i);

if(exp.charAt(i)=='{' ||exp.charAt(i)=='(' ||exp.charAt(i)=='[' ){

st.push(ch);

}

else{

if(isClosingParanthesis(ch)){

if(st.isEmpty()){

tag=1;

break;

}

else{

if(ch==']' && st.peek()=='['){

st.pop();

}

else{

if(ch=='}' && st.peek()=='{'){

st.pop();

}

else{

if(ch==')' && st.peek()=='('){

st.pop();

}

else{

tag=1;

}

}

}

}

}

}

}

return tag==0 && st.isEmpty()==true;

}

public static String adjustParanthesis(StringBuilder s){

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

char ch=s.charAt(i);

if(ch=='{' || ch=='['){

s.setCharAt(i, '(');

}

else{

if(ch=='}' || ch==']'){

s.setCharAt(i, ')');

}

}

}

return s.toString();

}

public static boolean isLogicalOperator(char ch){

return ch=='-' || ch == '^' || ch == '|' || ch == '@' || ch == '~' || ch=='+';

}

public static boolean isVariable(char ch){

return ch>='A' && ch<='Z';

}

public static boolean isOpeningParanthesis(char ch){

return (ch=='(' || ch=='{' || ch=='[');

}

public static boolean isClosingParanthesis(char ch){

return (ch==')'|| ch=='}' || ch==']');

}

public static boolean isAdjacentCharacter(String s){

char ch1='%',ch2;

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

ch2=s.charAt(i);

if(isVariable(ch1)&& isVariable(ch2)){

return false;

}

ch1=ch2;

}

return true;

}

public static boolean hasHigherPrecedence(char a,char b){

double value1=getOperatorValue(a);

double value2=getOperatorValue(b);

if(value1>value2){

return true;

}

else{

if(value1<value2){

return false;

}

else{

if(value1==value2){

return true;

}

}

}

return false;

}

public static int findIndex(char[] vars,char ch){

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

if(vars[i]==ch){

return i;

}

}

return 0;

}

public static double getOperatorValue(char x){

double value;

switch(x){

case '-':

value=1;

break;

case '^':

value=2;

break;

case '|':

value=3;

break;

case '+':

value=3;

break;

case '@':

value=4;

break;

case '~':

value = 5;

break;

default:

value=0;

}

return value;

}

private static int[][] printTruthTable(int n,char[] a) {

rows = (int) Math.pow(2,n);

columns = n;

int[][] array =new int[rows][n\*10];

int k,i,j,l,m;

for (i=0; i<rows; i++) {

k=0;

for (j=n-1; j>=0; j--) {

array[i][k]=(i/(int) Math.pow(2, j))%2;

k++;

}

}

System.out.println("\n\n----Initial Truth Table----\n\n");

for(i=0;i<columns;i++){

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

}

System.out.println("");

for(l=0;l<rows;l++){

for(m=0;m<columns;m++){

if(array[l][m]==0){

System.out.print("F ");

}

else{

System.out.print("T ");

}

}

System.out.println("");

}

return array;

}

public static int evaluatingTheExpression(String s,int[][] array,char[] var){

Stack<Character> s1=new Stack<>();

char result='%';

char operand2,operand1;

char ch;

TwoDArray=array;

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

ch=s.charAt(i);

if(isVariable(ch)){

s1.push(ch);

}

else{

if(isLogicalOperator(ch)){

if(isBinaryOperator(ch)){

if(s1.size()>=2){

operand2=s1.pop();

operand1=s1.pop();

int u=findIndex(var,operand1);

int v=findIndex(var,operand2);

result=printResult(var,ch,u,v);

s1.push(result);

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

result=0;

return result;

}

}

else{

if(s1.size()>=1){

operand1=s1.pop();

int u=findIndex(var,operand1);

result=printResult(var,ch,u);

s1.push(result);

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

result=0;

return result;

}

}

}

else{

System.out.println("\n\nInvalid Expression.\n\nProgram Ended Gracefully.\n\n");

result=0;

return result;

}

}

}

if(columns>=1){

System.out.println("--------COMPLETE TRUTH TABLE OF GIVEN EXPRESSION--------\n\n");

print2DArray(var);

if(isTautology(TwoDArray)){

System.out.println("\n\nConclusion : It's a Tautology.\n");

finalPrinting();

}

else{

if(isContradiction(TwoDArray)){

System.out.println("\n\nConclusion : It's a Contradiction.\n\n");

finalPrinting();

}

else{

System.out.println("\n\nConclusion : It's a Contingency.\n\n");

finalPrinting();

}

}

}

return result;

}

private static char printResult(char[] var,char c,int u,int v){

int[] extraArray;

char f;

switch(c){

case '^':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = "+var[u]+" ^ "+var[v]+"\n");

extraArray=AND(TwoDArray,var,u,v);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

case '|':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = "+var[u]+" | "+var[v]+"\n");

extraArray=OR(TwoDArray,var,u,v);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

case '@':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = "+var[u]+" --> "+var[v]+"\n");

extraArray=implication(TwoDArray,var,u,v);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

case '~':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = "+var[u]+" <--> "+var[v]+"\n");

extraArray=biConditional(TwoDArray,var,u,v);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

case '+':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = "+var[u]+" XOR "+var[v]+"\n");

extraArray=XOR(TwoDArray,var,u,v);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

default:

f='%';

}

return f;

}

private static char printResult(char[] var,char c,int u){

int[] extraArray ;

char f;

switch(c){

case '-':

System.out.println("\n---------------\nSTEP NO : " +stepNo+"\n---------------\n");

stepNo++;

System.out.println("Resultant variable : "+ascii+ " = NOT "+var[u]+"\n");

extraArray=negation(TwoDArray,var,u);

TwoDArray=adjust2DArray(TwoDArray,extraArray);

var=adjustVariablesArray(var);

f=var[columns];

columns++;

break;

default:

f='%';

}

return f;

}

private static int[] OR(int array[][],char var[],int u,int v){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]==0 && array[i][v]==0){

System.out.println(var[u] +" OR "+var[v] + " = F ");

newArray[k]=0;

}

else{

System.out.println(var[u] +" OR "+var[v] + " = T ");

newArray[k]=1;

}

k++;

}

System.out.println("");

return newArray;

}

private static int[] AND(int array[][],char var[],int u,int v){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]==1 && array[i][v]==1){

System.out.println(var[u] +" AND "+var[v] + " = T ");

newArray[k]=1;

}

else{

System.out.println(var[u] +" AND "+var[v] + " = F ");

newArray[k]=0;

}

k++;

}

System.out.println("");

return newArray;

}

private static int[] negation(int array[][],char var[],int u){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]==0){

System.out.println("NOT "+var[u]+" = T ");

newArray[k]=1;

}

else{

System.out.println("NOT "+var[u]+" = F ");

newArray[k]=0;

}

k++;

}

System.out.println("");

return newArray;

}

private static int[] implication(int array[][],char var[],int u,int v){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]==1 && array[i][v]==0){

System.out.println(var[u] +" --> "+var[v] + " = F ");

newArray[k]=0;

}

else{

System.out.println(var[u] +" --> "+var[v] + " = T ");

newArray[k]=1;

}

k++;

}

System.out.println("");

return newArray;

}

private static int[] biConditional(int array[][],char var[],int u,int v){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]==array[i][v]){

System.out.println(var[u] +" <--> "+var[v] + " = T ");

newArray[k]=1;

}

else{

System.out.println(var[u] +" <--> "+var[v] + " = F ");

newArray[k]=0;

}

k++;

}

System.out.println("");

return newArray;

}

private static int[] XOR(int array[][],char var[],int u,int v){

int[] newArray=new int[rows];

int k=0;

for(int i=0;i<rows;i++){

if(array[i][u]!=array[i][v]){

System.out.println(var[u] +" XOR "+var[v] + " = T ");

newArray[k]=1;

}

else{

System.out.println(var[u] +" XOR "+var[v] + " = F ");

newArray[k]=0;

}

k++;

}

System.out.println("");

return newArray;

}

public static int[][] adjust2DArray(int[][] array,int[] extraArray){

for(int i=0;i<rows;i++){

array[i][columns]=extraArray[i];

}

return array;

}

public static char[] adjustVariablesArray(char[] var){

var[columns]=ascii;

ascii++;

return var;

}

public static boolean isTautology(int[][] array){

int tag=1;

for(int i=0;i<rows;i++){

if(TwoDArray[i][columns-1]==0){

tag=0;

}

}

return tag==1;

}

public static boolean isContradiction(int[][] array){

int tag=0;

for(int i=0;i<rows;i++){

if(TwoDArray[i][columns-1]==1){

tag=1;

}

}

return tag==0;

}

public static boolean isUnaryOperator(char ch){

return ch=='-';

}

public static boolean isBinaryOperator(char ch){

return ch=='^' || ch=='@' || ch=='|' || ch=='~' ||ch=='+';

}

public static void print2DArray(char[] var){

int i,l,m;

for(i=0;i<columns;i++){

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

}

System.out.println("");

for(l=0;l<rows;l++){

for(m=0;m<columns;m++){

if(TwoDArray[l][m]==0){

System.out.print("F ");

}

else{

System.out.print("T ");

}

}

System.out.println("");

}

}

public static void finalPrinting(){

System.out.println("Total number of steps involved : "+(stepNo-1)+"\n\n\nProgram Ended Gracefully.\n\n\n");

}

}

• A program that generates a power set of a set

package powerset;

import java.util.\*;

public class PowerSet {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.println("\n\n----------------INSTRUCTIONS----------------\n\n1.Program can handle input of a set in comma seperated format with surrounded by Curly Brackets.\n"+

"\nE.g : {5,6,37,12,45} OR {a,bc,d}\n\n"+

"3.No other Brackets are allowed except These Curly Braces.\n\n");

System.out.print("Enter a set: ");

String in = input.nextLine();

if(checkValidity(in)){

in = in.replaceAll(" ", "");

in=in.replaceAll("\\{", "").replaceAll("\\}","");

String [] set = in.split(",");

System.out.println(Arrays.toString(set));

System.out.println("");

PowerSet(set, set.length);

System.out.println("\n\nPower Set Generated Successfully.\n\nProgram Ended Gracefully.\n\n");

}

else{

System.out.println("\n\nInvalid Input.\n\nProgram Ended Gracefully.\n\n");

}

}

public static void PowerSet(String[] set, int set\_size){

if (set\_size == 0){

System.out.println(" { { } }");

return;

}

int power = (int) Math.pow(2, set\_size);

int i, j;

System.out.print("{ ");

for(i = 0; i < power; i++){

System.out.print("{ ");

for(j = 0; j < set\_size; j++){

if((i & (1 << j)) != 0){

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

}

}

if(i==power-1){

System.out.print("} ");

}

else{

System.out.print("} , ");

}

}

System.out.print(" }");

System.out.println("");

}

public static boolean checkValidity(String exp){

int i,tag=0;

char ch;

Stack <Character>st=new Stack<>();

for(i=0;i<exp.length();i++){

ch=exp.charAt(i);

if(isOpeningParanthesis(ch)){

st.push(ch);

}

else{

if(isClosingParanthesis(ch)){

if(st.isEmpty()){

tag=1;

break;

}

else{

if(ch=='}' && st.peek()=='{'){

st.pop();

}

else{

tag=1;

break;

}

}

}

}

}

return tag==0 && st.isEmpty()==true;

}

public static boolean isOpeningParanthesis(char ch){

return (ch=='(' || ch=='{' || ch=='[');

}

public static boolean isClosingParanthesis(char ch){

return (ch==')'|| ch=='}' || ch==']');

}

}