class Result {

/\*

\* Complete the 'sExpression' function below.

\*

\* The function is expected to return a STRING.

\* The function accepts STRING nodes as parameter.

\*/

public static String sExpression(String nodes) {

// Write your code here

boolean[][] graph = new boolean [26][26];

HashSet<Character> node = new HashSet<Character>();

boolean E2 = false;

for(int i = 1; i < nodes.length(); i += 6){

int x = nodes.charAt(i)-'A', y = nodes.charAt(i+2)-'A';

if(graph[x][y])

E2 = true;

graph[x][y] = true;

node.add(nodes.charAt(i));

node.add(nodes.charAt(i+2));

}

boolean E1 = false;

for(int i=0;i<26;i++){

int count = 0;

for(int j=0;j<26;j++){

if(graph[i][j])

count++;

}

if(count>2)

return "E1";

}

if(E2) return "E2";

int numOfRoots = 0;

char root =' ';

for(char nod : node){

for(int i=0;i<26;i++){

if(graph[i][nod-'A'])

break;

if(i==25){

numOfRoots++;

root = nod;

boolean[] visited = new boolean[26];

if(IsCycle(nod, graph, visited))

return "E3";

}

}

}

if(numOfRoots==0) return "E3";

if(numOfRoots>1) return "E4";

if(root==' ') return "E5";

return GetExpressionHelper(root, graph);

}

private static boolean IsCycle(char node, boolean[][] graph, boolean[] visited){

if(visited[node-'A'])

return true;

visited[node-'A'] = true;

for(int i = 0; i < 26; i++){

if(graph[node-'A'][i]){

if(IsCycle((char)(i+'A'), graph, visited))

return true;

}

}

return false;

}

private static String GetExpressionHelper(char root, boolean[][] graph){

String left = "", right = "";

for(int i=0;i<26;i++){

if(graph[root-'A'][i]){

left = GetExpressionHelper((char)(i+'A'), graph);

for(int j=i+1;j<26;j++){

if(graph[root-'A'][j]){

right = GetExpressionHelper((char)(j+'A') ,graph);

break;

}

}

break;

}

}

return "("+root+left+right+")";

}

}