PROBLEMS: UNIT 5

BACKTRACKING

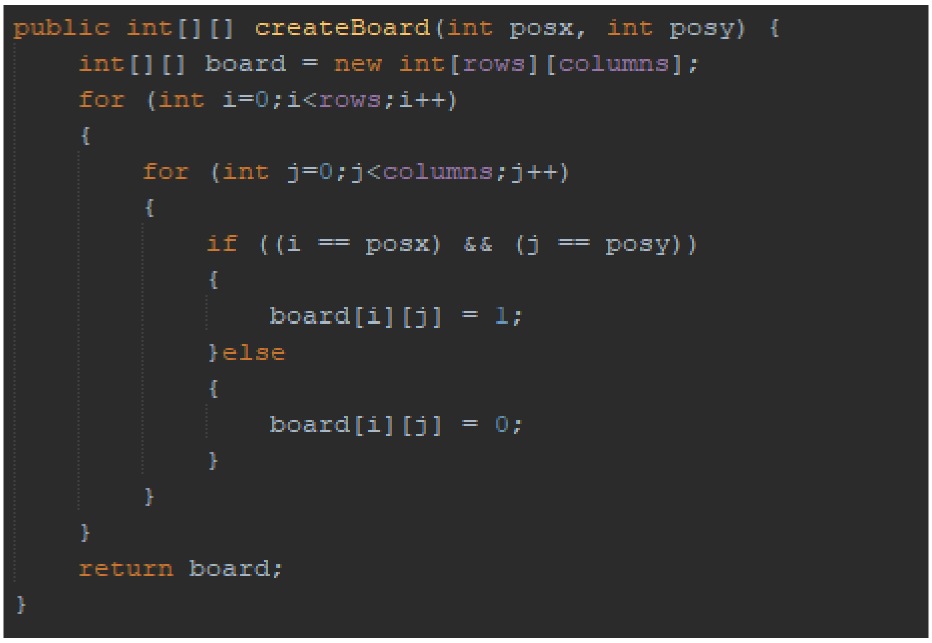
JAVIER GARCÍA JIMÉNEZ AND ISABEL MARTÍNEZ GÓMEZ

PROBLEM 4

In this problem, we are asked to implement a backtracking algorithm where we have to find the way in which the horse in a chess board, move around the board so that each position of the board is used only once in the course.

As we know that the 8x8 board always has a solution regardless of where the horse start, we have decided to give the board this dimension.

First, we have created the board in which the horse will be placed in a given position with coordinates posx and posy. We will place the horse giving it value 1 and initializing the rest of the board with value 0. We do this with the function createBoard:



```
public int[][] createBoard(int posx, int posy) {
    int[][] board = new int[rows][columns];
    for (int i=0;i<rows;i++)
    {
        for (int j=0;j<columns;j++)
        {
            if ((i == posx) && (j == posy))
            {
                board[i][j] = 1;
            }else
            {
                board[i][j] = 0;
            }
        }
    }
    return board;
}
```

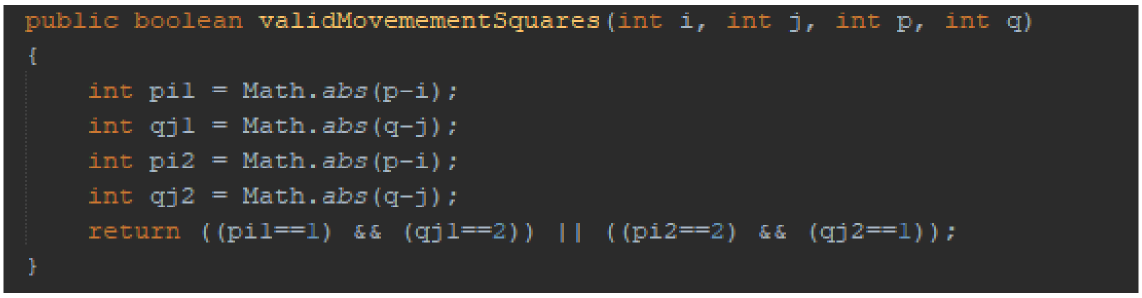
A horse on a chess board can perform eight possible movements. Therefore, a movement between one position and another is only valid if:

- (|p-i| = 1) && (|q-j| = 2)
- (|p-i| = 2) && (|q-j| = 1)

That is, a horse can only perform a movement by advancing two positions in either direction vertically and one position horizontally or by advancing two positions in either direction horizontally and one position vertically as long as it doesn't exceed the board limits.

To know if a movement is valid, we have created the validMovementSquare function in which we return true in case the movement is valid and false otherwise.

Our function look like this:

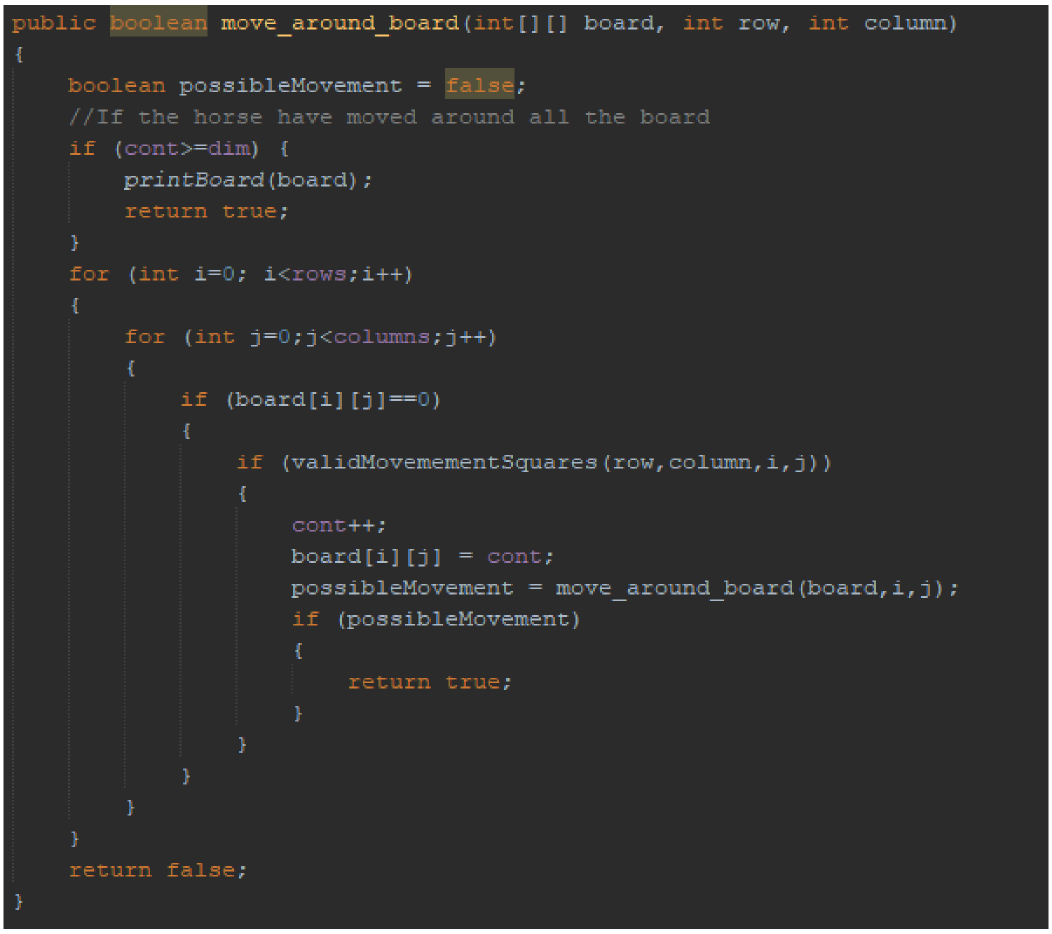


```java
public boolean validMovemementSquares(int i, int j, int p, int q)
{
    int pi1 = Math.abs(p-i);
    int qj1 = Math.abs(q-j);
    int pi2 = Math.abs(p-i);
    int qj2 = Math.abs(q-j);
    return ((pi1==1) && (qj1==2)) || ((pi2==2) && (qj2==1));
}
```

Now I will explain how we have implemented the backtracking algorithm so that the horse moves around the board, so that it does not repeat the position it has reached.

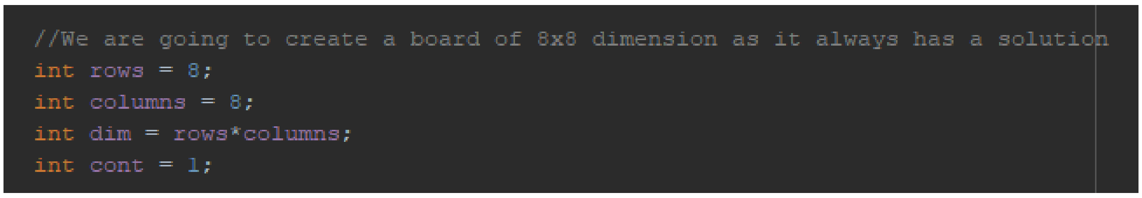
First, we check if the cont is greater or equal than the dimension of the board. If it is, we print the board and return true, so the backtracking algorithm will end. If it is not, we go through the chess board trying to move the horse to positions of the board. In case that the position that we are looking of the board is a 0, it means that previously the horse has not been moved to that square, so we can see if it can make any movements from that square.

As we mentioned before, the function that told us if a movement was valid is validMovementSquares at to which we pass the current position of the horse and the position to which we want to move the horse. If it is a valid movement, we increment the cont, we we assign the value of cont to the position where we move the horse and we recursively call the function now with that position so that we could find more positions on the board to go to from this new box.

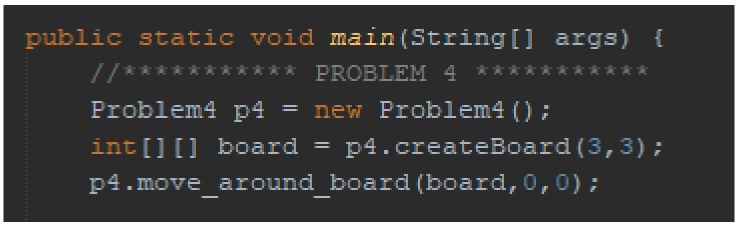


```java
public boolean move_around_board(int[][] board, int row, int column)
{
    boolean possibleMovement = false;
    //If the horse have moved around all the board
    if (cont>=dim) {
        printBoard(board);
        return true;
    }
    for (int i=0; i<rows;i++)
    {
        for (int j=0;j<columns;j++)
        {
            if (board[i][j]==0)
            {
                if (validMovemementSquares(row,column,i,j))
                {
                    cont++;
                    board[i][j] = cont;
                    possibleMovement = move_around_board(board,i,j);
                    if (possibleMovement)
                    {
                        return true;
                    }
                }
            }
        }
    }
    return false;
}
```

As we have commented at the beginning of the explanation, we have made an 8x8 dimension board so that we can see how the horse, initially in the middle position, has managed to move to all the squares on the board once. This are the initial values of the rows and columns and also we attached the main function:

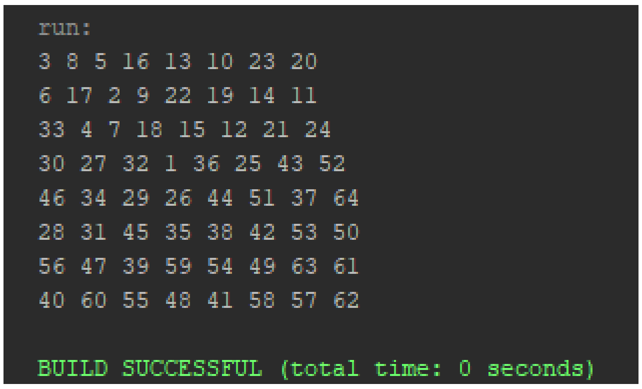


```java
//We are going to create a board of 8x8 dimension as it always has a solution
int rows = 8;
int columns = 8;
int dim = rows*columns;
int cont = 1;
```



```java
public static void main(String[] args) {
    //*********** PROBLEM 4 ***********
    Problem4 p4 = new Problem4();
    int[][] board = p4.createBoard(3,3);
    p4.move_around_board(board,0,0);
```

If we run the main, we obtain:



```
run:
3 8 5 16 13 10 23 20
6 17 2 9 22 19 14 11
33 4 7 18 15 12 21 24
30 27 32 1 36 25 43 52
46 34 29 26 44 51 37 64
28 31 45 35 38 42 53 50
56 47 39 59 54 49 63 61
40 60 55 48 41 58 57 62

BUILD SUCCESSFUL (total time: 0 seconds)
```

The numbers indicate the iteration in which the horse has moved to that square. As we see in the main, we create the board with the horse in the middle of the board, that is, in the box (3,3) → **1** , so, in the result, the horse is in the middle of the array, in the position (3,3).

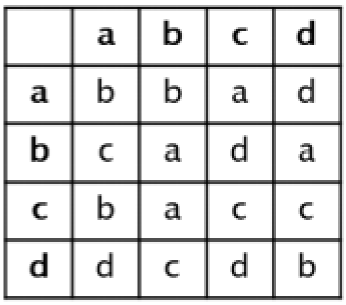
Then, the horse goes to the box (1,2) → **2** which means that the horse has moved two boxes horizontally to the left and one vertically up to get to that box. After that, the horse has moved to the box (0,0) → **3** and to get to that box, the horse has moved two boxes horizontally to the left and one vertically up and so on until all the horse has move around the board and placed in each box once.

If we had tested the execution with another dimension, perhaps the horse would not have been able to move through all the squares on the board. In this case, the squares in which the horse could not have reached through its movements will have a value of 0.

PROBLEM 6

In this problem, we are asked to implement a backtracking algorithm that, starting from a string of text and using the information stored in a substitution table M, is able to find a way to make the substitutions that allow reducing the text string to a final character, if possible.

The table M of substitutions is here:



	a	b	c	d
a	b	b	a	d
b	c	a	d	a
c	b	a	c	c
d	d	c	d	b

In order to realize the backtracking in a proper way, we need to consider a few thinks:

- If the size of the string is 1, its means that the string have been substituted into one character. If that character is out goal, the algorithm has found a proper way to substitute our initial string to our final goal, so we return true. If not, we will return false.

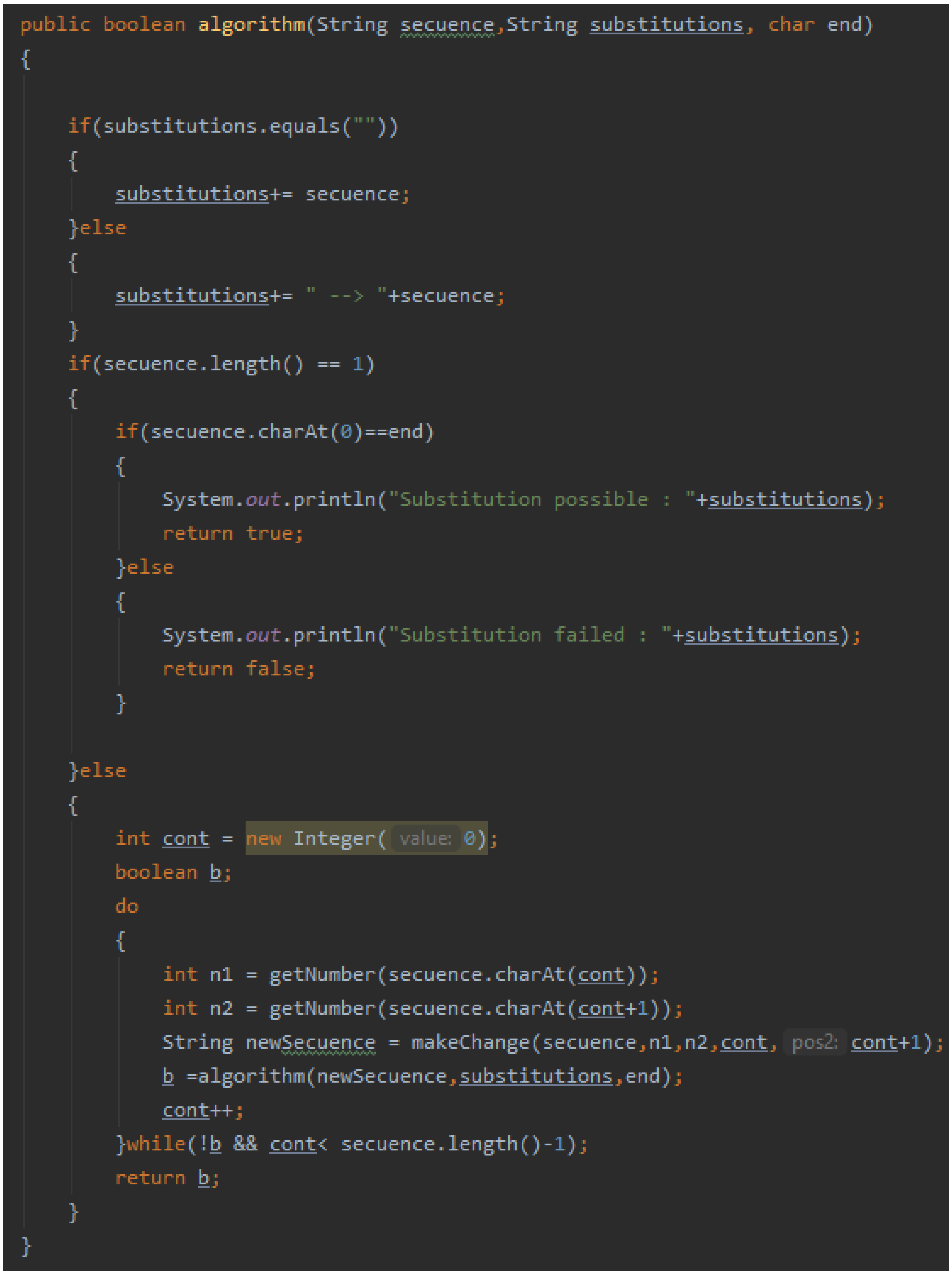
If the size is not 1, it means that we have still work to do. We will do this in order to create a backtracking algorithm. We need to loop through all the possible substitutions in our string. This is easy because for a string that has N length, there will be N-1 substitutions, because the substitutions are made of 2 letters. So for each string, there are string.length() -1 substitutions.

These provide us all the possible substitutions in our string, so we need to call recursively the function until we get a true value (we achieved our goal) or we receive a false value, so that means that that substitution does not arrive at a valid solution.

Taking all of this in account, out algorithm starts making substitutions and going through our string using backtracking until a solution is found.

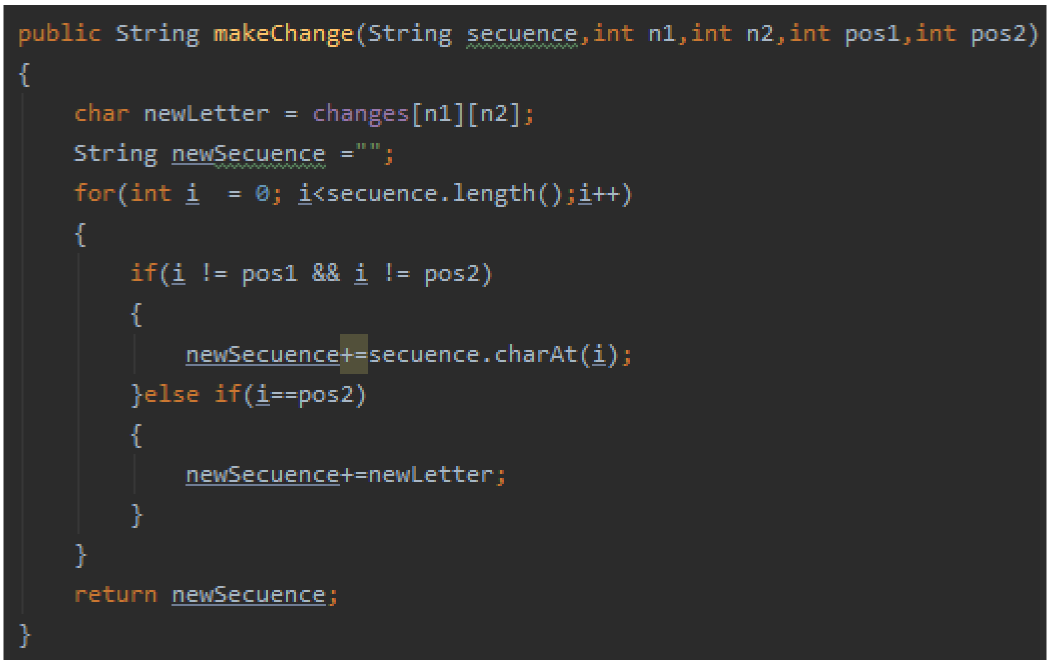
In order to provide de user some information about what substitutions have been made, we take also in account the substitutions with a string. Having that, we print on the screen the work of the program.

Our function looks like this:



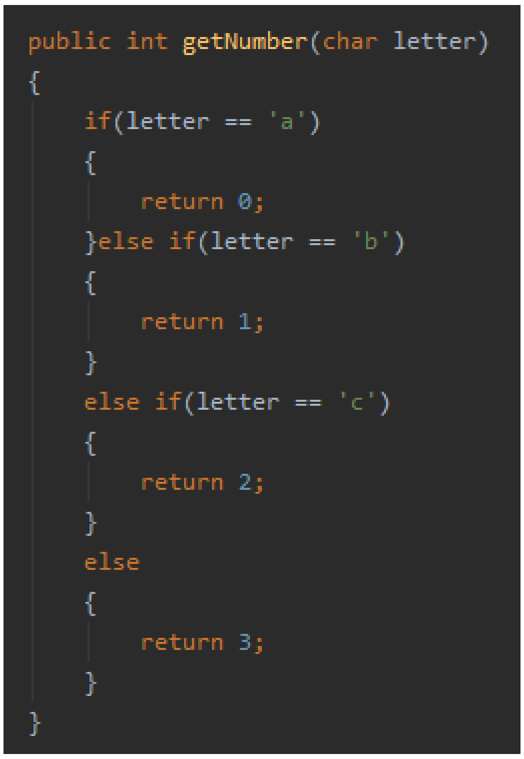
```java
public boolean algorithm(String secuence,String substitutions, char end)
{

    if(substitutions.equals(""))
    {
        substitutions+= secuence;
    }else
    {
        substitutions+= " --> "+secuence;
    }
    if(secuence.length() == 1)
    {
        if(secuence.charAt(0)==end)
        {
            System.out.println("Substitution possible : "+substitutions);
            return true;
        }else
        {
            System.out.println("Substitution failed : "+substitutions);
            return false;
        }

    }else
    {
        int cont = new Integer( value: 0);
        boolean b;
        do
        {
            int n1 = getNumber(secuence.charAt(cont));
            int n2 = getNumber(secuence.charAt(cont+1));
            String newSecuence = makeChange(secuence,n1,n2,cont, pos2: cont+1);
            b =algorithm(newSecuence,substitutions,end);
            cont++;
        }while(!b && cont< secuence.length()-1);
        return b;
    }
}
```

We also have a function called make change, that receives a string, and returns the string after doing a substitution:



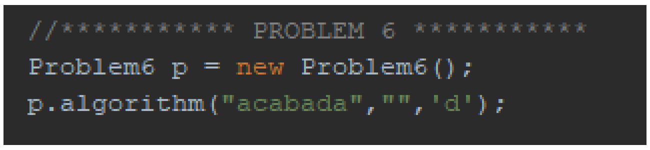
```
public String makeChange(String secuence,int n1,int n2,int pos1,int pos2)
{
    char newLetter = changes[n1][n2];
    String newSecuence ="";
    for(int i  = 0; i<secuence.length();i++)
    {
        if(i != pos1 && i != pos2)
        {
            newSecuence+=secuence.charAt(i);
        }else if(i==pos2)
        {
            newSecuence+=newLetter;
        }
    }
    return newSecuence;
}
```

We have also another function whose goal is to translate the letters into numbers, in order to manage the matrix of changes in a proper way. With this function, we assign one number to each letter, so the matrix can be managed with numbers:



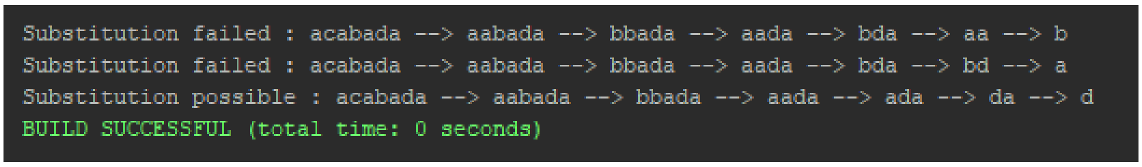
```
public int getNumber(char letter)
{
    if(letter == 'a')
    {
        return 0;
    }else if(letter == 'b')
    {
        return 1;
    }
    else if(letter == 'c')
    {
        return 2;
    }
    else
    {
        return 3;
    }
}
```

This is the main function



```
//*********** PROBLEM 6 ***********
Problem6 p = new Problem6();
p.algorithm("acabada","",'d');
```

If we run it, we get the following result:



```
Substitution failed : acabada --> aabada --> bbada --> aada --> bda --> aa --> b
Substitution failed : acabada --> aabada --> bbada --> aada --> bda --> bd --> a
Substitution possible : acabada --> aabada --> bbada --> aada --> ada --> da --> d
BUILD SUCCESSFUL (total time: 0 seconds)
```

As we can see, two substitutions fail while the last substitution ends in d and with a successful substitution.