UNIVERSIDADE ESTADUAL DO PARANÁ

CIÊNCIA DA COMPUTAÇÃO

GUILHERME HENRIQUE DE SOUZA

SARA SAORI SATAKE

DERIVAÇÃO DE GRAMÁTICAS REGULARES

CÓDIGO FONTE

APUCARANA – PR

2017

GUILHERME HENRIQUE DE SOUZA

SARA SAORI SATAKE

DERIVAÇÃO DE GRAMÁTICAS REGULARES

CÓDIGO FONTE

Código fonte do trabalho sobre Derivação de Gramáticas Regulares, apresentada na disciplina de Linguagens Formais Autômatos para o 2º ano de Ciência da Computação.

APUCARANA –PR

2017

```
package trabalholfa;


import java.util.Scanner;


public class TrabalhoLFA {
 public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StringBuilder resultados = new StringBuilder();


        System.out.println("====================================");
        System.out.println("\t  TRABALHO LFA\nDERIVAÇÃO DE GRAMÁTICAS REGULARES");
        System.out.println("Alunos: Guilherme Henrique de Souza \n\tSara Saori Satake");
        System.out.println("\n       UNESPAR - Apucarana\n\t      2017");
        System.out.println("====================================");


        int gld = 0;
        String ver;
        int ax = 0, check6 = 0;
        // Nessa parte o usuário digita a opção que ele deseja, apesar de os códigos poderem ser reaproveitados,
        // não optamos por essa hipótese. Toda parte dentro do if se repete no else, alterando somente algumas
        // funcionalidades especiais.


        System.out.println("Digite 0 para GLUE ou 1 para GLUD");
        ver = sc.next();


        for(int i = 0 ; i < ver.length(); i++){
            if(Character.isDigit(ver.charAt(i))){
```

```
            ax++;
        }
        if(ver.length() == ax && (Integer.parseInt(ver) == 0 ||
Integer.parseInt(ver) == 1)){
            check6++;
            gld = Integer.parseInt(ver);
        }
    }
    ax = 0;

    while(check6 !=1 || (Integer.parseInt(ver) != 0 && Integer.parseInt(ver) !=
1)){
        ax = 0;
        check6 = 0;
        System.out.println("Digite 0 para GLUE ou 1 para GLUD");
        ver = sc.next();

        for(int i = 0 ; i < ver.length(); i++){
        if(Character.isDigit(ver.charAt(i))){
            ax++;
        }
        if(ver.length() == ax && (ax == 0 || ax ==1)){
            check6++;
            gld = Integer.parseInt(ver);
        }
    }
    }
    ax = 0;
    if(gld == 1){

    // Foi instanciada as variáveis de verificação nessa parte.
```

```
        int qtdInstrucao = 0,parar = 0, indice = 0,qtdV = 0, qtdT = 0, auxnoinicial =
0;
        int check1 = 0, check2 = 0, check3 = 0, check4 = 0, check5 = 0, aux = 0;

        String verificacao;

        String vetorV[];
        vetorV = new String[999];

        String vetorT[];
        vetorT = new String[999];

        String Instrucao[];
        Instrucao = new String[9999];

        String palavra;

        String temporario[];
        temporario = new String[9999];

        String resultadoShow[];
        resultadoShow = new String[9999];

        // O noinicial é iniciado com S, mas logo abaixo ele já é reescrito pelo
símbolo que o usuário desejar.

        String noinicial = "S";

        // Para ter um melhor controle, a quantidade de variáveis é definida logo
no início.
```

```
        System.out.println("Digite a quantidade de variáveis (V): ");
        verificacao = sc.next();
        check4 = 0;

        // Nessa parte também é verificado se a quantidade de variáveis é um
numero.

        for(int i = 0; i < verificacao.length(); i++){
            if(Character.isDigit(verificacao.charAt(i)) == true){
                aux++;
            }
            if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
                check4++;
                qtdV = Integer.parseInt(verificacao);
            }
        }
        aux = 0;

            while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
                System.out.println("Digite apenas números inteiros > 0!");

                verificacao = sc.next();
                for(int i = 0; i < verificacao.length(); i++){
                    if(Character.isDigit(verificacao.charAt(i)) == true){
                    aux++;
                }
                    if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
                    check4++;
                    qtdV = Integer.parseInt(verificacao);
                    break;
```

```
        }
                aux = 0;
    }
        }
        aux = 0;
        check4 = 0;

    // Nessa parte começa a entrada e a validação das variáveis (V).

    for(int i = 1; i < qtdV + 1; i++){
        System.out.println("Digite a "+i+"ª variável: ");
        verificacao = sc.next();

        while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toUpperCase()) ||
Character.isAlphabetic(verificacao.charAt(0)) == false){
            System.out.println("Digite novamente: ");
            verificacao = sc.next();
        }

        vetorV[i] = verificacao;



        for(int j = 1; j < i; j++){
            if(vetorV[j].equals(vetorV[i])){
               aux++;
            }

            while(aux != 0){
                aux = 0;
```

```
                System.out.println("Váriável já sendo usada! Digite novamente: ");
                verificacao = sc.next();

                    while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toUpperCase()) ||
Character.isAlphabetic(verificacao.charAt(0)) == false){
                    System.out.println("Digite novamente: ");
                    verificacao = sc.next();
                    }

                        vetorV[i] = verificacao;

                for(int k = 1; k < i; k++){
                if(vetorV[k].equals(vetorV[i])){
               aux++;
            }
            }
          }
        }

       }
         aux = 0;

       // A Quantidade de alfabeto da linguagem também é solicitada e verificada
como visto abaixo.

       System.out.println("Digite a quantidade de alfabeto da linguagem (T): ");
       verificacao = sc.next();
       check4 = 0;
       for(int i = 0; i < verificacao.length(); i++){
          if(Character.isDigit(verificacao.charAt(i)) == true){
              aux++;
```

```
            }
            if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
                check4++;
                qtdT = Integer.parseInt(verificacao);
            }
        }
        aux = 0;

            while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
                System.out.println("Digite apenas números inteiros > 0!");

                verificacao = sc.next();
                for(int i = 0; i < verificacao.length(); i++){
                    if(Character.isDigit(verificacao.charAt(i)) == true){
                    aux++;
                }
                    if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
                    check4++;
                    qtdT = Integer.parseInt(verificacao);
                    break;
            }
                    aux = 0;
        }
             }
             aux = 0;
             check4 = 0;

        for(int i = 1; i < qtdT + 1; i++){

            // Entrada e validação dos caracteres do alfabeto.
```

```
        System.out.println("Digite o "+i+"º caractere do alfabeto: ");
        verificacao = sc.next();

        while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toLowerCase())){
            System.out.println("Digite novamente: ");
            verificacao = sc.next();
        }

        vetorT[i] = verificacao;

        for(int j = 1; j < i; j++){
            if(vetorT[j].equals(vetorT[i])){
                aux++;
            }
        }
            for(int l = 1; l < qtdV + 1; l++){
                 if(vetorV[l].equals(vetorT[i])){
                      aux++;
                   }
                }

            while(aux != 0){
              aux = 0;
              System.out.println("Váriável já sendo usada! Digite novamente: ");
              verificacao = sc.next();

                    while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toLowerCase())){
                    System.out.println("Digite novamente: ");
```

```
                verificacao = sc.next();
                }

              vetorT[i] = verificacao;

            for(int k = 1; k < i; k++){
              if(vetorT[k].equals(vetorT[i])){
              aux++;
              }
            }
            for(int m = 1; m < qtdV + 1; m++){
              if(vetorV[m].equals(vetorT[i])){
              aux++;
              }
            }
          }

}
aux = 0;

// Nessa parte o S é inserido, mas antes ele passa por uma
// verificação para ver se pode existir.

while(auxnoinicial == 0){
System.out.println("Digite o simbolo de partida (S): ");
noinicial = sc.next();
for(int i = 1; i < qtdV + 1; i++){
    if(noinicial.equals(vetorV[i])){
        auxnoinicial++;
    }
```

```
        }
        }

        // O resultadoShow índice 0 recebe o nó inicial, para fazer as futuras
        // comparações (Ele só é usado para imprimir os passos no final da
execução).

        resultadoShow[0] = noinicial;

        // A quantidade de ordens de produção também é solicitada e
armazenada.

        System.out.println("Digite a quantidade de ordens de produções: ");
        verificacao = sc.next();
        check4 = 0;
        for(int i = 0; i < verificacao.length(); i++){
            if(Character.isDigit(verificacao.charAt(i)) == true){
                aux++;
            }
            if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
                check4++;
                qtdInstrucao = Integer.parseInt(verificacao);
            }
        }
        aux = 0;

            while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
                System.out.println("Digite apenas números inteiros > 0!");
                verificacao = sc.next();
                for(int i = 0; i < verificacao.length(); i++){
                    if(Character.isDigit(verificacao.charAt(i)) == true){
```

```
                aux++;
            }
                if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
                check4++;
                qtdInstrucao = Integer.parseInt(verificacao);
                break;
          }
                aux = 0;
       }


          }
           aux = 0;
          check4 = 0;


       // As ordens de produção começam a ser inseridas,verificadas e então
armazenadas sequencialmente.
       System.out.println("-------------------------------------------------");
       System.out.println("=> Digite no padrão A>bC");
       System.out.println("- Sendo A e C pertencentes a V e b pertencente a T");
       System.out.println("- Caso tenha algum branco, simbolizar com *");
       System.out.println("-------------------------------------------------");
       for(int i = 1; i < qtdInstrucao + 1; i++){
          System.out.println("Digite a " + i + "ª ordem de produção: ");
          verificacao = sc.next();
          while(verificacao.length() != 4){
             System.out.println("Digite novamente a ordem de produção: ");
             verificacao = sc.next();
          }
             char[] verificachar = verificacao.toCharArray();
             for(int k = 1; k < qtdV + 1; k++){
```

```
                if(Character.toString(verificachar[0]).equals(vetorV[k])){
                   check1 ++;
                   }
                if(Character.toString(verificachar[3]).equals(vetorV[k])){
                    check2++;
                }
               }

               for(int s = 1; s < qtdT + 1; s++){
                   if(Character.toString(verificachar[2]).equals(vetorT[s])){
                       check3++;
                   }
               }

                   if(Character.toString(verificachar[2]).equals("*")){
                       check3++;
                   }

                   if(Character.toString(verificachar[3]).equals("*")){
                       check2++;
                   }

                   if(Character.toString(verificachar[1]).equals(">")){
                     check5++;
                   }
                   if(check1 == 1 && check2 == 1 && check3 == 1 && check5 == 1
&& verificacao.length() == 4){
                   Instrucao[i] = verificacao;
                   }
```

```
while(check1 != 1 || check2 != 1 || check3 != 1 || check5 != 1 || verificacao.length() != 4){
System.out.println("Ordem de produção inválida! Digite uma nova ordem de produção: ");
check1 = 0;
check2 = 0;
check3 = 0;
check5 = 0;
verificacao = sc.next();

char[] verificachar2 = verificacao.toCharArray();
for(int k = 1; k < qtdV + 1; k++){
if(Character.toString(verificachar2[0]).equals(vetorV[k])){
check1 ++;
}
if(Character.toString(verificachar2[3]).equals(vetorV[k])){
 check2++;
}
}

for(int s = 1; s < qtdT + 1; s++){
if(Character.toString(verificachar2[2]).equals(vetorT[s])){
    check3++;
}
}
if(Character.toString(verificachar2[2]).equals("*")){
    check3++;
}

if(Character.toString(verificachar2[3]).equals("*")){
    check2++;
```

```
                }
                if(Character.toString(verificachar2[1]).equals(">")){
                   check5++;
                }
                if(check1 == 1 && check2 == 1 && check3 == 1 && check5 == 1
&& verificacao.length() == 4){
                Instrucao[i] = verificacao;
                }
                }
                check1 = 0;
                check2 = 0;
                check3 = 0;
                check5 = 0;



        }
        int j = 0;


        // Nessa parte a ordem de produção é dividada em 3 partes, cada parte é
armazenada
        // em um índice do array temporario[].
        // P.e: S>aA --> temporario[1] = S, temporario[2] = a, temporari[3] = A.


        for(int i = 1; i< qtdInstrucao + 1; i++){
        char[] letras = Instrucao[i].toCharArray();
        j++;
        temporario[j] = Character.toString(letras[0]);
        j++;
        temporario[j] = Character.toString(letras[2]);
        j++;
        temporario[j] = Character.toString(letras[3]);
```

```
        }

        int gigante = 0;

        // Essa é a parte que começa a recursividade, sempre que uma palavra for testada
        // ele volta para essa parte, sendo possível testar várias palavras com o mesmo
        // "VTPS".

        while(gigante != 1){

        // A palavra é armazenada temporariamente na String palavra. Logo após
        // o armazenamento temos uma verificação para ver se a palavra contém
        // alguma das variáveis. Caso não tenha, a palavra já é recusada, economizando
        // tempo e processamento.

        boolean palavraver = false;
        int palavraveri = 0;
        palavra = "NULL";
        while(palavraver == false){
        System.out.println("Digite a palavra: ");
        palavra = sc.next();

        char[] charverific = palavra.toCharArray();
        for(int u = 0; u < palavra.length(); u++){
        for(int i = 0; i < qtdV + 1; i++){
            if(Character.toString(charverific[u]).equals(vetorV[i])){
                palavraveri++;
            }
        }
```

```
        }
        if(palavraveri == 0){
            palavraver = true;
            break;
        }
        else{
            System.out.println("==============================");
            System.out.println("= Essa palavra não pertence! =");
            System.out.println("==============================");
        }
        palavraveri = 0;
        }

        // Um array de Char chamado verific recebe palavra armazenando em
cada índice uma
        // letra de palavra.
        // P.e: abc --> verific[0] = a, verific[1] = b, verific[2].

        char[] verific = palavra.toCharArray();
        char[] verifi;
        verifi = new char[9999];

        // Foi passado para um array de char de tamanho 9999 e o restante dos
indíces
        // em branco foi completado com o "branco padrão", o "*".
        // P.e: abc --> abc*...*.

        for(int i = 0; i < palavra.length(); i++){
            verifi[i] = verific[i];
        }
```

```
        for(int i = palavra.length(); i < 9999; i++){
            verifi[i] = '*';
        }

        j = 0;

        // Nessa parte foram criadas e instanciadas várias variáveis para auxiliar
no
        // controle e transições dos "nós".

        String[] resultado;
        resultado = new String[9999];

        int[][] possibilidades;
        possibilidades = new int[9999][9999];

        int[][] backpassos;
        backpassos = new int[9999][9999];

        int passos[];
        passos = new int[9999];

        String[] backp;
        backp = new String[9999];

        int[] guardaz;
        guardaz = new int[9999];

        int[] guardax;
        guardax = new int[9999];
```

```
        int[] guardaindice;
        guardaindice = new int[9999];

        int[] guardat;
        guardat = new int[9999];

        for(int i = 0; i < 999; i++){
        resultado[i] = "*";
        }
        int[] poss;
        poss = new int[999];

        // Aqui o resultado índice 0 recebe o nó inicial, fazendo assim com que
        // a parte principal do algoritmo esteja finalmente pronta para rodar.

        resultado[0] = noinicial;
        int z = 0,x = 0, ok = 0, t = 0,cont = 0, auxcont = 0, back = 0, possi = 0, g =
0;
        int firstx = 0, indicepalavra = 0;
        boolean accept = false;
        boolean backtime = false;

        // Nessa parte o algoritmo começa a operar, ele só para quando acaba
toda a interação,
        // tanto com um resultado positivo ou negativo.

        while(parar != 1){

            // Esse laço abaixo serve para pegar o último índice escrito, ou seja,
            // aquele que não possui o "branco padrão".
```

```
        for(int i = 0; !"*".equals(resultado[i]) && resultado[i] != null; i++){
            indice = i;
        }

        // Com a posse desse "ultímo índice", são feitas comparações para saber
        // quantas possibilidades são possíveis.
        // As comparações consistem em verificar se o "ultímo índice" do resultado
        // bate com o temporário múltiplo de 3 (3,6,9,12...), pois como o temporário
        // é armazenado na forma X>xX o primeiro índice sempre vai ser o "estado"
        // dele, além dessa comparação, ele verifica se o indice da palavra bate com
        // o temporário indíce + 1 (P.e: S>aA --> bate com o temporario[i] no índice "S" e com
        // o temporário[i + 1] que no caso seria o "a"), caso os dois batam, ele armazena em
        // um array bidemensional de possibilidade, que a cada interação positiva, o x é aumentado
        // e o z se mantém no mesmo índice, só aumentando quando ele avance ou diminua uma "altura do nó",
        // o Z representaria a "altura do nó" e "x", as possibilidades.
        // P.e: O nó na altura Z = 2 possui X = 3 possibilidades.
        // Além dessas duas verificações temos outras duas que seriam para tratar quando temos uma ordem
        // de produção indo para outro "estado" deixando o vazio.
        // P.e: S>A.
        // Nesse caso ele faz a mesma verifação que acontece anteriormente, só que acrescida de outras duas.
        // Uma que verifica se o temporari[i+1] é igual ao "branco padrão" e se a próxima é diferente do
        // "branco padrão", caso seja verdadeira, ele armazena na possibilidade do mesmo jeito que visto acima.
```

```
        // Outra parte importante do algortimo é a parte do "BackTime", que
toda vez que não é encontrado uma "saída",

        // ele volta para as últimas configurações válidas e não passa por esse
processo abaixo, pois ele já tem todas

        // as possibilidades possíveis armazenadas.


        if(backtime == false){

        for(int i = 1; i < qtdInstrucao * 3; i += 3){

            if((resultado[indice].equals(temporario[i]) && verifi[indicepalavra] ==
temporario[i+1].charAt(0) && (!(temporario[i].equals(temporario[i+2]) &&
temporario[i+1].equals("*")))) || ((resultado[indice].equals(temporario[i]) &&
temporario[i+1].equals("*") && !temporario[i+2].equals("*")) &&
(!(temporario[i].equals(temporario[i+2]) && temporario[i+1].equals("*")))) ){

            possibilidades[z][x] = i;

            // Caso a possibilidade seja 0, ele admite que a primeira tentativa do
algoritmo será pelo índice 0.

            if(possi == 0){

                firstx = x;

            }

            possi++;

            x++;

            }

        }

        x = firstx;


        // Também é armazenada na poss[z] o índice de possibilidades, ou seja,

        // é possível saber quantas possibilidades temos na "altura" da arvóre.


        poss[z] = possi;


        }


        // Caso o backtime esteja ativo, ele verifica o último índice e recebe o
```

```
        // ultímo índice da palavra da altura em que o nó se encontra; neste
        // caso ele também verifica a quantidade de possibilidades para fazer
        // as futuras "transições".

        if(backtime == true){
            for(int i = 0; !"*".equals(resultado[i]) && resultado[i] != null; i++){
            indice = i;
            }
            indicepalavra = guardaindice[z];
            for(int i = 1; i < qtdInstrucao * 3; i += 3){
            if(resultado[indice].equals(temporario[i])  && verifi[indicepalavra] ==
temporario[i+1].charAt(0)){
                possi++;
            }
        }
        }

        // Caso a possibilidade seja um ele verifica se a possibilidade não está
no padrão S>*A,
        // se ele estiver, o resultado "último índice" recebe o temporário "índice
2". No nosso exemplo
        // ele receberia o A e o índice da palavra receberia -1, pois nesse caso
só foi alterado um
        // índice do resultado.
        // No caso dele não entrar na possibilidade acima, ele só troca o "último
índice" e o "último
        // índice + 1".
        // P.e¹: S>*A --> S é trocado por A.
        // P.e²: S:aA --> S é trocado por aA.
        // O backtime é sempre desligado após essas trocas.

        if(possi == 1){
```

```
            if(temporario[possibilidades[z][x] + 1].equals("*") &&
!temporario[possibilidades[z][x] + 2].equals("*")){
            resultado[indice] = temporario[possibilidades[z][x] + 2];
            indicepalavra--;
            }
            else{
            resultado[indice] = temporario[possibilidades[z][x] + 1];
            resultado[indice + 1] = temporario[possibilidades[z][x] + 2];
            }
            backtime = false;
        }

        // Caso a possibilidade seja maior que 1, ele verifica se o backtime
        // também está ativo, caso não esteja ele faz um "backup" do resultado
        // em todos seus índices, para isso ele concatena todos os índices
        // do resultado usando o StringBuilder. Após esse "backup" ser
        // concatenado, ele armazena o "backup' na "altura" do nó.
        // Após essa verificação, ele executa novamente os mesmos passos
citados acima.

        else if(possi > 1){
            if(possi > 1 && backtime == false){
            for(int i = 0; !"*".equals(resultado[i]) && resultado[i] != null; i++){
              resultados.append(resultado[i]);
            }
            backp[z] = resultados.toString();

            // Também deletamos todo o conteúdo do resultado,
            // podendo assim, ser reaproveitado.

            resultados.delete(0,9999);
```

```
                guardaz[g] = z;
                guardaindice[z] = indicepalavra;
                g++;
                }
            backtime = false;
            if(temporario[possibilidades[z][x] + 1].equals("*") &&
!temporario[possibilidades[z][x] + 2].equals("*")){
                resultado[indice] = temporario[possibilidades[z][x] + 2];
                indicepalavra--;
            }
            else{
            if( !"*".equals(temporario[possibilidades[z][x] + 1]) &&
temporario[possibilidades[z][x] + 1] != null){
            resultado[indice] = temporario[possibilidades[z][x] + 1];
            }
            if( !"*".equals(temporario[possibilidades[z][x] + 2]) &&
temporario[possibilidades[z][x] + 1] != null){
            resultado[indice + 1] = temporario[possibilidades[z][x] + 2];
            }

            // A única diferença seria essas duas verifcações a mais, caso o
temporário esteja recebendo
            // um "branco padrão". Se estiver, ele adiciona o "branco padrão" no
array resultado.

            if( "*".equals(temporario[possibilidades[z][x] + 1]) ||
temporario[possibilidades[z][x] + 1] == null){
            resultado[indice] = "*";
            }
            if( "*".equals(temporario[possibilidades[z][x] + 2]) ||
temporario[possibilidades[z][x] + 1] == null){
            resultado[indice + 1] = "*";
            }
```

```
        }
        }

        // Nesta parte temos um laço for usado para comparar o índice
        // de dois arrays, o resultado e o verifi (palavra digitada),
        // toda vez que os índices batem, ele soma um na váriavel OK.

        for(int i = 0; i < palavra.length(); i++){
            if(Character.toString(verifi[i]).equals(resultado[i])){
                ok++;
            }
        }

        // Aqui é verificado se o último índice da palavra bate com o último
índice do reultado.

        if(!"*".equals(resultado[palavra.length()]) && resultado[palavra.length()]
!= null){
            if(Character.toString(verifi[palavra.length() -
1]).equals(resultado[palavra.length()])){
            ok++;
        }
        }

        // Na parte abaixo fazemos a subtração do índice até chegar em 1,
        // dessa forma conseguimos o número da instrução e, por
consequência,
        // qual instrução foi usada.
        // Assim que é feito a verifação, ela é armazenada num array chamado
        // passos[].
        // Caso a quantidade de possibilidades for igual a 1, quer dizer
        // que a instrução utilizada foi a 1, não precisando fazer nenhuma
```

```
// operação.

auxcont++;
cont = possibilidades[z][x];
if(possibilidades[z][x] > 1){
    while(parar != 1){
        cont = cont - 3;
        auxcont++;
        if(cont == 1){
            passos[t] = auxcont;
            break;
        }
    }
}
else{
    passos[t] = 1;
}

// Caso a possibilidade de troca seja maior que 1, ele faz um
// "backup" de passos para poder voltar, assim como acontece com
// o resultado.

if(possi > 1){
    for(int i = 0; i <= t; i ++){
    backpassos[z][i] = passos[i];
    }
    guardat[z] = t;
}
t++;
```

```
        //  Aqui é feito as verifações. Caso ok seja igual ao tamanho da palavra, a palavra foi aceita
        // e o laço é parado, fazendo com que seja impresso as instruções passadas pelo algoritmo.
        // Caso o "último índice" do resultado seja diferente do "branco padrão", ele necessita
        // ter um ok a mais para ser aceito, pois como visto acima ele passa por uma verifação
        // que adiciona um ok a mais.

        if(!"*".equals(resultado[palavra.length()]) && resultado[palavra.length()] != null){
        if(ok == (palavra.length()) + 1){
            System.out.println("==================");
            System.out.println("= Palavra Aceita =");
            System.out.println("==================");
            accept = true;
            break;
        }
        }
        else{
          if(ok == (palavra.length())){
            System.out.println("==================");
            System.out.println("= Palavra Aceita =");
            System.out.println("==================");
            accept = true;
            break;
        }
        }

        // Nesta parte é contada quantos índices do resultado são diferentes
        // do "branco padrão".
```

```
        for(int i = 0; i < 999; i++){
            if(!"*".equals(resultado[i])){
                back++;
            }
        }

        // O x é armazenado no array guardax, sempre levando em conta
        // a altura do "nó".

        guardax[z] = x;

        // Caso o back seja maior que o tamanho da palavra + 1, ou a
quantidade
        // de possibilidades seja igual a 0, ele entra neste if.
        // A função dele se baseia em retornar no "nó" anterior que tem mais
        // de duas possibilidades e que não tenha sido visitada.
        // Limitamos a "altura" do no, dessa forma ele nunca fica em um loop
        // infinito, mas em contra partida temos um limite maximo de 500 de
        // altura.

        if(back > (palavra.length() + 1) || possi == 0 || z > 500){

            // Os passos passados pelo algortimo são zerados usando o número
            // 999 para representar um passo "nulo".

            for(int i = 0; i < 9999; i ++){
                passos[i] = 999;
            }

            // Caso o g seja 0, quer dizer que não temos "backup" para retornar,
```

```
// fazendo com que a palavra não seja aceita.

if(g == 0){
    accept = false;
    break;
}

// O z recebe o que foi guardado anteriormente;
// como o g sempre soma um para pegarmos o antigo
// valor, temos que diminuir 1.
// Neste caso o nosso "nó" recebe o que estava guardado,
// portanto tem uma nova "altura".

z = guardaz[g - 1];

// Se esse Z for 0 quer dizer que não temos mais alturas
// na árvore, fazendo com que o algoritmo pare.

if(z < 0){
    accept = false;
    break;
}

// Nessa parte verificamos se o X é menor que a
// poss[] do mesmo, ou seja, se a quantidade
// de possibilidades é maior que o índice do x.
// P.e: Possibilidade do nó na altura 3 é 5, o x tem
// que ser menor que 5; se for maior ele é obrigado
// a descer uma altura do nó.
```

```
if(guardax[z] < (poss[z])){

// Nessa parte preenchemos o array passos com o "backup",
// este é o array que armazena todos as ordens de
// produções passadas até então.

for(int i = 0; i < guardat[z]; i++){
  passos[i] = backpassos[z][i];
}

// T também recebe o guardat na altura que o
// "nó" se encontra.

t = guardat[z];

// A variavel char go recebe o "backup" da altura
// do novo nó e armazena em varíos índices do
// array.

char[] go = backp[z].toCharArray();

// Todos os índices do array resultado são
// marcados com o "branco padrão".

for(int i = 0; i < 999; i++){
    resultado[i] = "*";
}

// O array resultado então, recebe o "backup".
```

```
for(int i = 0; i < backp[z].length(); i++){
    resultado[i] = Character.toString(go[i]);
}

// X recebe o que estava guardado na altura
// antiga do nó e soma mais um, ou seja,
// ele pega a última possibilidade que
// o antigo nó percorreu e vai para o
// próximo.

x = guardax[z];
x++;

}

// Caso todas as possibilidades daquele nó
// já tenham sido percorridas, ele volta mais
// um nó, e enquanto não acha um nó com
// possibilidades possíveis, ele vai voltando.
// Caso o z chegue a -1, o algoritmo para
// e sabe que a palavra não é possível.
// A parte restante do else é igual ao do if
// tendo como sua principal diferença que ele
// é responsável por fazer a "volta" do nó.

else{
z--;
if(z < 0){
    accept = false;
    break;
```

```
}
x = guardax[z];
while(possibilidades[z][x] == 0){
z--;
if(z < 0){
    accept = false;
    break;
}
    x = guardax[z];
}
if(z < 0){
    accept = false;
    break;
}
for(int i = 0; i < guardat[z]; i++){
    passos[i] = backpassos[z][i];
}
t = guardat[z];
x++;
if(backp[z] == null){
    accept = false;
    break;
}
char[] go = backp[z].toCharArray();
for(int i = 0; i < 999; i++){
    resultado[i] = "*";
}
for(int i = 0; i < backp[z].length(); i++){
    resultado[i] = Character.toString(go[i]);
}
```

```
        }

        // Toda vez que ele tem que fazer algum tipo de
        // "backup", o backtime é setado para true.
        // Como visto acima o backtime restringe e autoriza
        // muitas parte do código.

        backtime = true;
    }

    // Nessa parte zeramos todas as váriaveis
    // auxiliares para recomeçar o ciclo.

    back = 0;
    ok = 0;
    auxcont = 0;
    possi = 0;

    // Se o backtime não estiver ativo o índice da palavra sobe um;
    // assim como a "altura" do nó, zerando também o X.
    // Uma nova altura sempre zera o x, fazendo com que ele sempre
    // siga a primeira possibilidade encontrada.

    if(backtime == false){
    indicepalavra++;
    z++;
    x = 0;
    }
}
```

```
// Caso o accept seja true a palavra é aceita,
// imprimindo assim todas as informações pertinentes
// ao usuário.

if(accept == true){
System.out.println("Ordens de produção passadas pelo algoritmo: ");

// O resultadoShow[] é preenchido com o "branco padrão".

for(int i = 1; i < 9999; i++){
    resultadoShow[i] = "*";
}
resultadoShow[0] = noinicial;
// Temos aqui um laço de repetição que imprime todos os passos
// que são diferentes do "branco do passos", ou seja, 9999.

for(int i = 0; i < t ; i++){
    if(passos[i] != 9999){
    System.out.println("("+passos[i] + ")" + Instrucao[passos[i]]);

    // Nesse caso optamos por mostrar na prática como seriam as
    // modificações sofridas, fazendo com que a instrução seja
    // repartida e alocada na variável passosfinal..

    char[] passosfinal = Instrucao[passos[i]].toCharArray();

    // Pegamos o "último índice" do resultadoShow para
    // fazermos a troca.

    for(int u = 0; !resultadoShow[u].equals("*"); u++){
```

```
            indice = u;
        }

        // Também tratamos o caso se tivermos a opção de
        // uma ordem de produção ir direta para outra,
        // deixando o vazio.
        // P.e: S>A, no algoritmo ele seria interpretado
        // como S>*A.

        if(Character.toString(passosfinal[2]).equals("*") &&
!Character.toString(passosfinal[3]).equals("*")){
        resultadoShow[indice] = Character.toString(passosfinal[3]);
        }
        else{
        resultadoShow[indice] = Character.toString(passosfinal[2]);
        resultadoShow[indice + 1] = Character.toString(passosfinal[3]);
        }

        // Imprimimos também o resultado após a interação acima.

        for(int u = 0; u < 9999; u++){
            if(!resultadoShow[u].equals("*")){
                System.out.print(resultadoShow[u]);
            }
        }
            System.out.println(" ");
        }
    }
    }

    // Caso o accept seja falso, é impresso uma
```

```
        // frase alertando que a palavra não é aceita.

        if(accept == false){
            System.out.println("==============================");
            System.out.println("= Essa palavra não pertence! =");
            System.out.println("==============================");
        }
    }
}
        // Daqui para baixo começa a parte do GLUE. O código se parece
        // muito o GLUD, tendo apenas algumas modificações no procedimento
        // de substituição.

        else{

        // Como toda a parte abaixo tem a mesma lógica que o GLUD,
        // será comentado somente quando houver alguma diferença.

        int qtdInstrucao = 0,parar = 0, indice = 0,qtdV = 0, qtdT = 0, auxnoinicial =
0, check1 = 0, check2 = 0, check3 = 0, check4 = 0, check5 = 0, aux = 0;

        String verificacao;

        String vetorV[];
        vetorV = new String[999];

        String vetorT[];
        vetorT = new String[999];

        String Instrucao[];
        Instrucao = new String[9999];
```

```
String palavra;

String temporario[];
temporario = new String[9999];

String noinicial = "S";

System.out.println("Digite a quantidade de variáveis (V): ");
verificacao = sc.next();
check4 = 0;
for(int i = 0; i < verificacao.length(); i++){
    if(Character.isDigit(verificacao.charAt(i)) == true){
        aux++;
    }
    if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
       check4++;
       qtdV = Integer.parseInt(verificacao);
    }
}
aux = 0;

    while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
        System.out.println("Digite apenas números inteiros > 0!");

        verificacao = sc.next();
        for(int i = 0; i < verificacao.length(); i++){
            if(Character.isDigit(verificacao.charAt(i)) == true){
            aux++;
        }
```

```
                if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
                check4++;
                qtdV = Integer.parseInt(verificacao);
                break;
            }
                aux = 0;
        }
         }
         aux = 0;
         check4 = 0;

        for(int i = 1; i < qtdV + 1; i++){
        System.out.println("Digite a "+i+"ª variável: ");
        verificacao = sc.next();

        while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toUpperCase()) ||
Character.isAlphabetic(verificacao.charAt(0)) == false){
            System.out.println("Digite novamente: ");
            verificacao = sc.next();
        }



        vetorV[i] = verificacao;



        for(int j = 1; j < i; j++){
            if(vetorV[j].equals(vetorV[i])){
               aux++;
            }
```

```
            while(aux != 0){
                aux = 0;
                System.out.println("Váriável já sendo usada! Digite novamente: ");
                verificacao = sc.next();

                while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toUpperCase()) ||
Character.isAlphabetic(verificacao.charAt(0)) == false){
                System.out.println("Digite novamente: ");
                verificacao = sc.next();
                }

                    vetorV[i] = verificacao;

                for(int k = 1; k < i; k++){
                if(vetorV[k].equals(vetorV[i])){
                aux++;
            }
            }
        }
    }

    }
    aux = 0;
    System.out.println("Digite a quantidade de alfabeto da linguagem (T): ");
    verificacao = sc.next();
    check4 = 0;
    for(int i = 0; i < verificacao.length(); i++){
        if(Character.isDigit(verificacao.charAt(i)) == true){
```

```
            aux++;
        }
        if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
          check4++;
          qtdT = Integer.parseInt(verificacao);
        }
      }
      aux = 0;

        while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
          System.out.println("Digite apenas números inteiros > 0!");

          verificacao = sc.next();
          for(int i = 0; i < verificacao.length(); i++){
              if(Character.isDigit(verificacao.charAt(i)) == true){
              aux++;
          }
              if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
              check4++;
              qtdT = Integer.parseInt(verificacao);
              break;
        }
              aux = 0;
      }
       }
       aux = 0;
       check4 = 0;

      for(int i = 1; i < qtdT + 1; i++){
        System.out.println("Digite o "+i+"º caractere do alfabeto: ");
```

```
        verificacao = sc.next();

        while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toLowerCase())){
            System.out.println("Digite novamente: ");
            verificacao = sc.next();
        }

        vetorT[i] = verificacao;

        for(int j = 1; j < i; j++){
            if(vetorT[j].equals(vetorT[i])){
                aux++;
            }
        }
            for(int l = 1; l < qtdV + 1; l++){
                 if(vetorV[l].equals(vetorT[i])){
                      aux++;
                   }
                }

            while(aux != 0){
              aux = 0;
              System.out.println("Váriável já sendo usada! Digite novamente: ");
              verificacao = sc.next();

              while(verificacao.length()!= 1 || verificacao.equals("*") ||
!verificacao.equals(verificacao.toLowerCase())){
              System.out.println("Digite novamente: ");
              verificacao = sc.next();
              }
```

```
            vetorT[i] = verificacao;

          for(int k = 1; k < i; k++){
            if(vetorT[k].equals(vetorT[i])){
            aux++;
            }
          }
          for(int m = 1; m < qtdV + 1; m++){
            if(vetorV[m].equals(vetorT[i])){
            aux++;
            }
          }
        }

}
aux = 0;

while(auxnoinicial == 0){
System.out.println("Digite o simbolo de partida (S): ");
noinicial = sc.next();
for(int i = 1; i < qtdV + 1; i++){
    if(noinicial.equals(vetorV[i])){
        auxnoinicial++;
    }
}
}



System.out.println("Digite a quantidade de ordens de produções: ");
```

```
        verificacao = sc.next();
        check4 = 0;
        for(int i = 0; i < verificacao.length(); i++){
            if(Character.isDigit(verificacao.charAt(i)) == true){
                aux++;
            }
            if((verificacao.length() == aux) && Integer.parseInt(verificacao) > 0){
                check4++;
                qtdInstrucao = Integer.parseInt(verificacao);
            }
        }
        aux = 0;

            while((check4 != 1) || Integer.parseInt(verificacao) <= 0){
                System.out.println("Digite apenas números inteiros > 0!");
                verificacao = sc.next();
                for(int i = 0; i < verificacao.length(); i++){
                    if(Character.isDigit(verificacao.charAt(i)) == true){
                    aux++;
                }
                    if((verificacao.length() == aux) && Integer.parseInt(verificacao) >
0){
                    check4++;
                    qtdInstrucao = Integer.parseInt(verificacao);
                    break;
            }
                    aux = 0;
        }

            }
             aux = 0;
```

```
        check4 = 0;


    // Aqui também é verificado se a instrução é permitida e tem seus
parâmetros
    // modificados para o padrão GLUE.
    System.out.println("-------------------------------------------------");
    System.out.println("=> Digite no padrão A>Bc");
    System.out.println("- Sendo A e B pertencentes a V e c pertencente a T");
    System.out.println("- Caso tenha algum branco, simbolizar com *");
    System.out.println("-------------------------------------------------");


    for(int i = 1; i < qtdInstrucao + 1; i++){
       System.out.println("Digite a " + i + "ª ordem de produção: ");
       verificacao = sc.next();
       while(verificacao.length() != 4){
          System.out.println("Digite novamente a ordem de produção: ");
          verificacao = sc.next();
       }


          char[] verificachar = verificacao.toCharArray();
          for(int k = 1; k < qtdV + 1; k++){
           if(Character.toString(verificachar[0]).equals(vetorV[k])){
              check1 ++;
             }
           if(Character.toString(verificachar[2]).equals(vetorV[k])){
               check2++;
           }
          }


          for(int s = 1; s < qtdT + 1; s++){
             if(Character.toString(verificachar[3]).equals(vetorT[s])){
```

```
                check3++;
            }
        }

            if(Character.toString(verificachar[2]).equals("*")){
                check2++;
            }

            if(Character.toString(verificachar[3]).equals("*")){
                check3++;
            }
            if(Character.toString(verificachar[1]).equals(">")){
              check5++;
            }
            if(check1 == 1 && check2 == 1 && check3 == 1 && check5 == 1
&& verificacao.length() == 4){
            Instrucao[i] = verificacao;
            }
            while(check1 != 1 || check2 != 1 || check3 != 1 || check5 != 1 ||
verificacao.length() != 4){
            System.out.println("Ordem de produção inválida! Digite uma nova
ordem de produção: ");
            check1 = 0;
            check2 = 0;
            check3 = 0;
            check5 = 0;
            verificacao = sc.next();

            char[] verificachar2 = verificacao.toCharArray();
            for(int k = 1; k < qtdV + 1; k++){
            if(Character.toString(verificachar2[0]).equals(vetorV[k])){
            check1 ++;
```

```
                }
                if(Character.toString(verificachar2[2]).equals(vetorV[k])){
                 check2++;
                }
                }

                for(int s = 1; s < qtdT + 1; s++){
                if(Character.toString(verificachar2[3]).equals(vetorT[s])){
                    check3++;
                }
                }
                if(Character.toString(verificachar2[2]).equals("*")){
                    check2++;
                }

                if(Character.toString(verificachar2[3]).equals("*")){
                    check3++;
                }
                if(Character.toString(verificachar2[1]).equals(">")){
                   check5++;
                }
                if(check1 == 1 && check2 == 1 && check3 == 1 && check5 == 1
&& verificacao.length() == 4){
                Instrucao[i] = verificacao;
                }
                }
                check1 = 0;
                check2 = 0;
                check3 = 0;
                check5 = 0;
```

```
}
int j = 0;
for(int i = 1; i< qtdInstrucao + 1; i++){
char[] letras = Instrucao[i].toCharArray();
j++;
temporario[j] = Character.toString(letras[0]);
j++;
temporario[j] = Character.toString(letras[2]);
j++;
temporario[j] = Character.toString(letras[3]);
}
int gigante = 0;

while(gigante != 1){
boolean palavraver = false;
int palavraveri = 0;
palavra = "NULL";
while(palavraver == false){
System.out.println("Digite a palavra: ");
palavra = sc.next();

char[] charverific = palavra.toCharArray();
for(int u = 0; u < palavra.length(); u++){
for(int i = 0; i < qtdV + 1; i++){
    if(Character.toString(charverific[u]).equals(vetorV[i])){
        palavraveri++;
    }
}
}
```

```
if(palavraveri == 0){
    palavraver = true;
}
else{
    System.out.println("==============================");
    System.out.println("= Essa palavra não pertence! =");
    System.out.println("==============================");
}
palavraveri = 0;
}
char[] verific = palavra.toCharArray();
char[] verifi;
verifi = new char[9999];
char[] verifitemp;
verifitemp = new char[9999];

// Nessa parte invertemos a palavra.
// Dessa forma, ganhamos uma melhora
// no desempenho do algoritmo, pois não temos que
// inverter a palavra em todo ciclo para comparar se
// ela foi aceita ou não.
// P.e: abc --> cba.

int auxverifi = palavra.length() - 1;
for(int i = 0;i < palavra.length(); i++){
    verifitemp[i] = verific[auxverifi];
    auxverifi--;
}
for(int i = 0; i < palavra.length(); i++){
    verifi[i] = verifitemp[i];
```

```
}

for(int i = palavra.length(); i < 9999; i++){
    verifi[i] = '*';
}

j = 0;

String[] resultado;
resultado = new String[9999];

String[] resultadoShow;
resultadoShow = new String[9999];
resultadoShow[0] = noinicial;

int[][] possibilidades;
possibilidades = new int[9999][9999];

int[][] backpassos;
backpassos = new int[9999][9999];

int passos[];
passos = new int[9999];

String[] backp;
backp = new String[9999];

int[] guardaz;
guardaz = new int[9999];
```

```
        int[] guardax;
        guardax = new int[9999];

        int[] guardaindice;
        guardaindice = new int[9999];

        int[] guardat;
        guardat = new int[9999];

        for(int i = 0; i < 9999; i++){
        resultado[i] = "*";
        }

        int[] poss;
        poss = new int[999];

        resultado[9998] = noinicial;
        int z = 0,x = 0, ok = 0, t = 0,cont = 0, auxcont = 0, back = 0, possi = 0, g =
0;
        int firstx = 0, indicepalavra = 0;
        boolean accept = false;
        boolean backtime = false;
        while(parar != 1){

            // Para pegarmos o "último índice" do GLUE, começamos de trás para
frente,
            // ou seja, ele tem que começar do índice 9998 e ir diminuindo.
            // P.e: *...*abc --> logo o último índice 9996.

            for(int i = 9998; !"*".equals(resultado[i]) && resultado[i] != null; i--){
                indice = i;
```

```
        }

        // A comparação também sofre uma leve mudança. Agora, ele compara com o
        // temporário índice[x+2].
        // P.e: S>Aa --> Ele comparia com índice 1(S) e o com o índice 3(a).
        // A verifacação de "branco padrão" também se mantém, mas também com uma
        // leve modificação nos índices. Agora, se o terceiro índice é branco e o
        // segundo não.
        // P.e: S>A*

        if(backtime == false){
        for(int i = 1; i < qtdInstrucao * 3; i += 3){
            if((resultado[indice].equals(temporario[i]) && verifi[indicepalavra] == temporario[i+2].charAt(0) && (!(temporario[i].equals(temporario[i+1]) && temporario[i+2].equals("*")))) || ((resultado[indice].equals(temporario[i]) && temporario[i+2].equals("*") && !temporario[i+1].equals("*")) && (!(temporario[i].equals(temporario[i+1]) && temporario[i+2].equals("*")))) ){
            possibilidades[z][x] = i;
            if(possi == 0){
                firstx = x;
            }
            possi++;
            x++;
            }
        }
        x = firstx;
        poss[z] = possi;
        }
        if(backtime == true){
```

```
            // Como descrito anteriormente, o índice também é recuperado da
mesma forma,
            // assim como a nova verificação de possibilidades..


            for(int i = 9998; !"*".equals(resultado[i]) && resultado[i] != null; i--){
            indice = i;
            }
            indicepalavra = guardaindice[z];
            for(int i = 1; i < qtdInstrucao * 3; i += 3){
            if(resultado[indice].equals(temporario[i])  && verifi[indicepalavra] ==
temporario[i+2].charAt(0)){
                possi++;
            }
          }
          }
          if(possi == 1){


            // A troca também sofreu uma leve modificação, fazendo com que o
resultado índice receba
            // a possibilidade + 2 e o resultado índice - 1 receba a possibilidade +
1.
            // P.e¹: S -- S>Aa --> Aa
            // P.e²: A -- A>Ba --> Baa.


            if(!temporario[possibilidades[z][x] + 1].equals("*") &&
temporario[possibilidades[z][x] + 2].equals("*")){
            resultado[indice] = temporario[possibilidades[z][x] + 1];
            indicepalavra--;
            }
            else{
            resultado[indice] = temporario[possibilidades[z][x] + 2];
            resultado[indice - 1] = temporario[possibilidades[z][x] + 1];
```

```
            }
            backtime = false;
        }


        else if(possi > 1){

            // O "backup" se manteve.

            if(possi > 1 && backtime == false){
            for(int i = 0; i < 9999; i++){
              if(!resultado[i].equals("*")){
              resultados.append(resultado[i]);
              }
            }
            backp[z] = resultados.toString();
            resultados.delete(0,9999);
            guardaz[g] = z;
            guardaindice[z] = indicepalavra;
            g++;
            }
        backtime = false;

        // Nesse caso, também temos a mesma modificação ocorrida acima,
        // fazendo com que ele sempre repita a troca na seguinte ordem:
        //          A>Ba          Aba                Baba
        // Índices: 1 23  / Índice palavra: 3  /    Índice palavra: 4

        if(!temporario[possibilidades[z][x] + 1].equals("*") &&
temporario[possibilidades[z][x] + 2].equals("*")){
            resultado[indice] = temporario[possibilidades[z][x] + 2];
```

```
            indicepalavra--;
        }
        else{
        if( !"*".equals(temporario[possibilidades[z][x] + 2]) &&
temporario[possibilidades[z][x] + 2] != null){
        resultado[indice] = temporario[possibilidades[z][x] + 2];
        }
        if( !"*".equals(temporario[possibilidades[z][x] + 1]) &&
temporario[possibilidades[z][x] + 1] != null){
        resultado[indice - 1] = temporario[possibilidades[z][x] + 1];
        }
        if( "*".equals(temporario[possibilidades[z][x] + 2]) ||
temporario[possibilidades[z][x] + 2] == null){
        resultado[indice] = "*";
        }
        if( "*".equals(temporario[possibilidades[z][x] + 1]) ||
temporario[possibilidades[z][x] + 1] == null){
        resultado[indice - 1] = "*";
        }
        }
        }
        j = 0;

        // Aqui usamos dois índices para fazer a verificação,
        // ele compara o índice 0 do verifi com o último. Logo
        // após essa verificação, o índice do resultado é diminuído,
        // e o índice do verifi, acrescido.
        // Dessa forma, conseguimos verificar se a derivação da palavra
        // está indo para o caminho certo.

        for(int i = 9998; i >= 0; i--){
            if(Character.toString(verifi[j]).equals(resultado[i])){
```

```
            ok++;
        }
        j++;
    }

    // As verificações a seguir são iguais ao GLUD só mudando os índices,
    // sempre trabalhando de trás para frente.
    // P.e: *...*abc
    //          <--

    if(!"*".equals(resultado[9998 - palavra.length()]) && resultado[9998 - palavra.length()] != null){
        if(Character.toString(verifi[palavra.length() - 1]).equals(resultado[9998 - palavra.length()])){
        ok++;
    }
    }
    auxcont++;
    cont = possibilidades[z][x];
    if(possibilidades[z][x] > 1){
        while(parar != 1){
            cont = cont - 3;
            auxcont++;
            if(cont == 1){
                passos[t] = auxcont;
                break;
            }
        }
    }
    else{
        passos[t] = 1;
```

```
            }
            if(possi > 1){
                for(int i = 0; i <= t; i ++){
                backpassos[z][i] = passos[i];
                }
                guardat[z] = t;
            }
            t++;

            if(!"*".equals(resultado[9998 - palavra.length()]) && resultado[9998 -
palavra.length()] != null){
            if(ok == 1000){
                System.out.println("==================");
                System.out.println("= Palavra Aceita =");
                System.out.println("==================");
                accept = true;
                break;
            }
            }
            else{
              if(ok == 9999){
                System.out.println("==================");
                System.out.println("= Palavra Aceita =");
                System.out.println("==================");
                accept = true;
                break;
            }
            }
            for(int i = 0; i < 9999; i++){
                if(!"*".equals(resultado[i])){
                    back++;
```

```
    }
}
guardax[z] = x;

// A única diferença na "volta do nó" é o
// jeito que o "backup" volta a ser armazenada no array resultado[].

if(back > (palavra.length() + 1) || possi == 0 || z > 500){
    for(int i = 0; i < 9999; i ++){
        passos[i] = 999;
    }
    if(g == 0){
        accept = false;
        break;
    }
    z = guardaz[g - 1];
    if(z < 0){
        accept = false;
        break;
    }
    if(guardax[z] <= (poss[z])){
    for(int i = 0; i < guardat[z]; i++){
      passos[i] = backpassos[z][i];
    }
    t = guardat[z];
    char[] go = backp[z].toCharArray();
    for(int i = 0; i < 9999; i++){
        resultado[i] = "*";
    }
```

```
// Na hora de recuperarmos o "backup", também tivemos
// que trabalhar com dois índices, mas sempre mantendo
// a mesma lógica de operação.

j = backp[z].length() - 1;
for(int i = 9998; i > (9998 - backp[z].length()); i--){
    resultado[i] = Character.toString(go[j]);
    j--;
}
x = guardax[z];
x++;
}
else{
z--;
if(z < 0){
    accept = false;
    break;
}
x = guardax[z];
while(possibilidades[z][x] == 0){
    z--;
    if(z < 0){
    accept = false;
    break;
}
    x = guardax[z];
}
if(z < 0){
    accept = false;
    break;
```

```
    }
    for(int i = 0; i < guardat[z]; i++){
        passos[i] = backpassos[z][i];
    }
    t = guardat[z];
    x++;
    if(backp[z] == null){
        accept = false;
        break;
    }
    char[] go = backp[z].toCharArray();
    for(int i = 0; i < 9999; i++){
        resultado[i] = "*";
    }
    j = backp[z].length() - 1;
    for(int i = 9998; i > (9998 - backp[z].length()); i--){
        resultado[i] = Character.toString(go[j]);
        j--;
    }
    }
    backtime = true;
}
back = 0;
ok = 0;
auxcont = 0;
possi = 0;
if(backtime == false){
indicepalavra++;
z++;
x = 0;
```

```
            }
        }

        // Assim que a palavra é aceita ou não, ela passa pelos mesmos procedimentos
        // do GLUD, com a diferença na maneira que os índices
        // são tratados.

        if(accept == true){
        System.out.println("Ordens de produção passadas pelo algoritmo: ");
        for(int i = 0; i < 9999; i++){
            resultadoShow[i] = "*";
        }
        resultadoShow[9998] = noinicial;
        for(int i = 0; i < t ; i++){
            if(passos[i] != 9999){
            System.out.println("("+passos[i] + ")"+Instrucao[passos[i]]);
            }
            char[] passosfinal = Instrucao[passos[i]].toCharArray();
            for(int u = 9998; !"*".equals(resultadoShow[u]) && resultadoShow[u] != null; u--){
                indice = u;
            }
            if(!Character.toString(passosfinal[2]).equals("*") && Character.toString(passosfinal[3]).equals("*")){
            resultadoShow[indice] = Character.toString(passosfinal[2]);
            }
            else{
            resultadoShow[indice] = Character.toString(passosfinal[3]);
            resultadoShow[indice - 1] = Character.toString(passosfinal[2]);
            }
            for(int u = 0; u < 9999; u++){
```

```
            if(!resultadoShow[u].equals("*")){
                System.out.print(resultadoShow[u]);
            }
        }
            System.out.println(" ");


    }
    }
    if(accept == false){
        System.out.println("==============================");
        System.out.println("= Essa palavra não pertence! =");
        System.out.println("==============================");
    }
    }
    }
 }
}
```