Oscar Gutierrez Godoy

A01635648

Proyecto de segundo parcial

Implementación del algoritmo CYK

El algoritmo CYK es un algoritmo que te permite saber si una cadena es aceptada por cierta gramática en la forma normal de Chomsky.

Algo que cabe mencionar antes de iniciar con la explicación del algoritmo CYK, es que creé una clase llamada Nodo, que sirve como elemento de entrada en la matriz. La razón por la que lo hice es para saber cuales son sus "hijos", en otras palabras, qué producciones se juntaron para generar esa producción en la matriz.

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

En otras palabras, si en esta matriz b se generó por a y d, b es la raíz del nodo y a y d son sus hijos.

```java
class Nodo{

    public String raiz;
    public Nodo hijoDerecho;
    public Nodo hijoIzquierdo;

    public Nodo(String raiz) {
        this.raiz = raiz;
    }

    public Nodo(String raiz, Nodo hijoDerecho, Nodo hijoIzquierdo) {
        this.raiz = raiz;
        this.hijoDerecho = hijoDerecho;
        this.hijoIzquierdo = hijoIzquierdo;
    }

    public boolean tieneHijos() {
        return hijoDerecho != null && hijoIzquierdo != null;
    }
}
```

La manera en la que el algoritmo funciona es que se crea una matriz con lado igual a la longitud de la palabra, y se pobla la diagonal principal con los símbolos generadores que lleven a los símbolos terminales de cada carácter de la palabra. En el proyecto esto fue realizado así:

```
for(int i = 0; i<MatrizCYK.length;i++) {
    for(ArrayList<String> Produccion : Gramatica) {
        if(Produccion.contains(Character.toString(cadena.charAt(i)))){
            MatrizCYK[i][i].add(new Nodo(Produccion.get(0)));
        }
    }
}
```

Después, para cada cuadro en cada diagonal superior a la diagonal principal, se tiene que calcular la casilla. Para calcular la casilla, tienes que obtener todos los símbolos a la izquierda y todos los símbolos debajo de dicha casilla, hasta llegar a la diagonal principal.

Entonces, iniciando por el símbolo de la diagonal principal, se concatena por el símbolo inmediatamente debajo de la casilla, y si existe una producción que genere la concatenación de los símbolos, se añade a la casilla, si no, no se agrega nada y continúa.

Esto se hace para cada casilla de la hasta que llegas a la inmediata a la izquierda y a la diagonal principal por abajo.

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

Por ejemplo, en esta matriz, para calcular c harías la concatenación de a con e, agregarías algo a c si es posible, y luego la concatenación de b y f, agregado también el resultado a c.

Esto se hizo con dos funciones auxiliares. La función resolverCasilla(int, int) se encarga de recorrer las casillas a la izquierda y abajo y agregar los resultados a la casilla deseada en la posición que se le pasó como parámetro.

```
public ArrayList<Nodo> resolverCasilla(int i, int j){
    //i = numero de fila
    //j = numero de columna
    int columna = i;
    ArrayList<Nodo> resultado = new ArrayList<Nodo>();
    for(int fila= i+1; fila<MatrizCYK.length;fila++) {
        //System.out.println(fila+""+columna);
        //columna ira disminuyendo para representar los cuadros a la izquierda, y fila aumentado para representar los cuadros abajo
        for(int x = 0; x<MatrizCYK[fila][j].size();x++) {
            for(int y = 0; y<MatrizCYK[i][columna].size();y++) {
                resultado.addAll(checar(MatrizCYK[i][columna].get(y),MatrizCYK[fila][j].get(x)));
            }
        }
        columna++;
    }
    return resultado;
}
```

Por otro lado, también se utilizó la función checar(String, String) que toma ambos string que se pasaron como parámetro, los concatena y checa en la gramática que producciones generan esa concatenación, si las hay. Regresa la lista de símbolos generadores de esas producciones, y regresa un String vacío si no existe.

```
public ArrayList<Nodo> checar(Nodo a, Nodo b) {
    ArrayList<Nodo> resultado = new ArrayList<Nodo>();
    String cadenaAChecar = a.raiz + b.raiz;
    for(int i=0; i<Gramatica.size();i++) {
        for(int j= 1; j < Gramatica.get(i).size(); j++) {
            if(Gramatica.get(i).get(j).equals(cadenaAChecar)) {
                resultado.add(new Nodo(Gramatica.get(i).get(0),a,b));
            }
        }
    }
    return resultado;
}
```

Entonces, usando esas dos funciones, podemos ir diagonal por diagonal resolviendo casillas y guardando los resultados, haciendo uso de programación dinámica.

```
for(int cont = 1; cont<MatrizCYK.length;cont++) {
    for(int i=0;i<MatrizCYK.length-cont;i++) {
        MatrizCYK[i][i+cont]=resolverCasilla(i,i+cont);
    }
}
```

Y por último, al final, lo único que tenemos que hacer es revisar si al final la casilla de hasta arriba a la derecha contiene el símbolo inicial de la gramática, S. Si lo contiene decimos que la cadena es aceptada, y si no, no es aceptada.

```
for(Nodo node : MatrizCYK[0][MatrizCYK.length-1]) {
    if(node.raiz.equals("S")){
        return true;
    }
}

return false;
```

Luego, para resolver el árbol de derivación, lo que hacemos es revisar recursivamente los hijos de cada nodo, que recordemos que son los Strings que concatenamos para obtener la raíz del nodo en cuestión. Lo hacemos de la siguiente manera:

```java
public void imprimirDerivacion() {
    for(Nodo node : MatrizCYK[0][MatrizCYK.length-1]) {
        if(node.raiz.equals("S")){
            imprimirNodo(node);
            break;
        }
    }
}

public void imprimirNodo(Nodo node) {
    if(node.tieneHijos()) {
        System.out.println(node.raiz);
        imprimirNodo(node.hijoDerecho);
        imprimirNodo(node.hijoIzquierdo);
    }else {
        System.out.println(node.raiz);
    }
}
```

Este proyecto me ayudó bastante a comprender mejor como funciona el algoritmo CYK. Hubo dos mayores complicaciones a la hora de hacer el proyecto. Una se presentó de la forma que no entendía bien el algoritmo. Yo creía que para resolver una casilla, había que concatenar primero las dos inmediatamente a la izquierda y abajo, e ir así hasta llegar a la diagonal.

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

En otras palabras, creía que para calcular c, las concatenaciones eran be y af, cuando en realidad son ae y bf. Esto generó un problema porque cadenas de prueba que si deberían ser aceptadas no lo eran, pero al imprimir la matriz el resultado era el esperado según mi entendimiento del algoritmo. Así que tuve que leer e investigar de nuevo para poder corregir el algoritmo con un funcionamiento correcto.

La otra dificultad se presentó por razones muy diferentes. Por motivos personales, tuve que cambiar la computadora donde estaba haciendo el proyecto, sin embargo se me había olvidado hacer un push de la última versión. Entonces, cuando quise resumir el proyecto habiendo terminado el problema personal por el que se pasó en la otra computadora, me di cuenta que una muy gran parte del proyecto no estaba disponible. Entonces me tuve que apurar y hacer todo rápido de nuevo. La memoria ayudó, pero aun así tuvo bastantes bugs a pesar de ser la segunda vez que lo hacía.