COCKE-KASAMI-YOUNGER ALGORITHM

A PYTHON IMPLEMENTATION

KOFFI ANDERSON KOFFI,
MATTHEW MILLS

University of Idaho

11/20/18

CONTENTS

INTRODUCTION

In this project, we will implement the Cocke-Kasami-Younger (CYK) Algorithm using the Python programming language. In fact, the CYK Algorithm is a membership algorithm for context-free grammars. Thus, using the CYK algorithm, it is possible to check whether a string is generated by the grammar of a context-free language.

IMPLEMENTATION

We implemented a version of the CYK algorithm using the python programming language and hosted the code source on github (see repository: https://github.com/kandersonko/cyk_algorithm). Our implementation required the grammar to be in Chomsky Normal Form. The version of the algorithm we implemented is the following:

CYK ($\mathcal{G}, w$)
    $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, $\Sigma \cup \mathcal{V} = \{\mathsf{X}_1, \ldots, \mathsf{X}_r\}$, $w = w_1 w_2 \ldots w_n$.
begin
  Initialize the 3d array $\mathsf{B}[1 \ldots n, 1 \ldots n, 1 \ldots r]$ to FALSE
  for $i = 1$ to n **do**
    for $(\mathsf{X}_j \to x) \in \mathcal{R}$ **do**
      if $x = w_i$ **then** $\mathsf{B}[i, i, j] \leftarrow$ TRUE.
  for $i = 2$ to n **do**   /* Length of span */
    for $L = 1$ to $n - i + 1$ **do**   /* Start of span */
      $R = L + i - 1$ /* Current span $s = w_L w_{L+1} \ldots w_R$ */
      for $M = L + 1$ to R **do**   /* Partition of span */
        /* $x = w_L w_{L+1} \ldots w_{M-1}$, $y = w_M w_{M+1} \ldots w_R$, and $s = xy$ */
        for $(\mathsf{X}_\alpha \to \mathsf{X}_\beta \mathsf{X}_\gamma) \in \mathcal{R}$ **do**
          /* Can we match X_β to x and X_γ to y? */
          if $\mathsf{B}[L, M-1, \beta]$ and $\mathsf{B}[M, R, \gamma]$ **then**
            $\mathsf{B}[L, R, \alpha] \leftarrow$ TRUE   /* If so, then can generate s by X_α! */
  for $i = 1$ to r **do**
    if $\mathsf{B}[1, n, i]$ **then return** TRUE
  return FALSE

Figure 1: The CYK algorithm.

Our code sources can be found in the appendix.

RESULTS

Using our implementation, we were able to check the membership of different string for the following grammar:

$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

Figure 2: Grammar G

We found that the string w="aabbb" is not in L(G), but the string w="aaabbb" is in L(G) (see figure 3).

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                1: python

Welcome to the CYK Interactive algorithm!
Enter the grammar in CNF form (E.g. S->AB|a;A->a;B->b): S->$|AB|XB;T->AB|XB;X->AT;A->a;B->b
Enter the string to check: aabbb
The string w="aabbb" is not in grammar G!
To continue enter "y" or press any key to exit: y
Enter the grammar in CNF form (E.g. S->AB|a;A->a;B->b): S->$|AB|XB;T->AB|XB;X->AT;A->a;B->b
Enter the string to check: aaabbb
The string w="aaabbb" is in grammar G!
To continue enter "y" or press any key to exit:
```

Figure 3: Output of an execution of the algorithm

This result agrees with our findings when we apply manually the algorithm.

This implementation can be downloaded on GitHub from the repository https://github.com/kandersonko/cyk_algorithm as a zip file and can be extracted into a folder "cyk_algorithm". Then, the code can be run on a computer with Python 3 installed by running on the following command on the command line "python main.py". Also, it is necessary to be in the folder containing "main.py". A simple "cd cyk_algorithm" should suffice to set the working directory to the folder "cyk_algorithm" containing the source code.

CONCLUSION

In this project, we implemented the CYK Algorithm in Python programming language. Our implementation can correctly check the membership of a string in the language generated by a grammar. The CYK Algorithm is important because it can be applied to check whether a keyword in a programming language.

APPENDIX

main.py

```python
from grammar import Grammar
from CYKAlgo import CYKAlgo


def main():

    print("Welcome to the CYK Interactive algorithm!")

    command = "y"
    while(command == "y"):
        grammar_text = input("Enter the grammar in CNF form (E.g. S->AB|a;A-
>a;B->b): ")
        if grammar_text == "": break
        G = Grammar(grammar_text.strip())

        w = input("Enter the string to check: ")

        cykAlgo = CYKAlgo(G)
        if (cykAlgo.membership(w.strip())):
            print("The string w=\"{}\" is in grammar G!".format(w))
        else:
            print("The string w=\"{}\" is not in grammar G!".format(w))

        command = input("To continue enter \"y\" or press any key to exit: ")
    print("Bye!")


if __name__ == '__main__':
    main()
```

gammar.py

```
class Grammar(object):

    def __init__(self, G):
        """ __init__ takes a string G
            and parses parses the productions into an array of productions
        """
        self.rules = G.split(';');
        self.productions = dict()
        for rule in self.rules:
            startVar = rule.split('->')[0]
            varSet = rule.split("->")[1]
            variables = [x for x in varSet.split('|') if x.isupper()]
            terminals = [x for x in varSet.split('|') if x.islower()]
            self.productions[startVar] = {"variables": variables,
"terminals": terminals}
```

CYKAlgo.py

```
class CYKAlgo:
    def __init__(self, G):
        """ initilizes with the grammar G
        """
        self.G = G

    def membership(self, w):
        B = dict()
        X = dict()
        V = [i for i in self.G.productions.keys()]
        for k,v in enumerate(self.G.productions.keys()):
            X[v]=k
        n = len(w)
        r = len(X)

        # case where S->a is the only production
        if(r==1 and w in self.G.productions[V[0]]["terminals"]):
            return True

        # initialize all items in B to false
        for i in range(n):
            for j in range(n):
                for k in range(r):
                    B[i, j, k] = False
```

```python
        # production A->a
        for i in range(n):
            for j,v in enumerate(X):
                if w[i] in self.G.productions[v]["terminals"]:
                    B[i, i, j] = True

        # production A -> BC
        for i in range(1, n):
            for L in range(n-i+1):
                R = L + i - 1
                for M in range(L+1, R):
                    for v in range(r):
                        P=self.G.productions[V[v]]
                        variables = P["variables"]
                        if(len(variables)):
                            b, c = tuple(variables[0])
                            s, t = X[b],X[c]
                            if(B[L, M-1, s] and B[M, R, t]):
                                B[L, R, v] = True

        r = n-1
        for i in range(r):
            if(B[r, n-1, i]):
                return True
        return False
```

REFERENCES:

"Lecture 15." *Ethics and Engineering*, courses.engr.illinois.edu/cs373/sp2009/lectures/.