ECOTE – Final project

Semester: 8

Author: Abhiraj Singh

Subject: Top-Down parser with backtracking

General overview and assumptions

The goal is to create a Top-Down parser with backtracking which can take input and parse that input through the tree from top to bottom and will backtrack at each step when the production doesn't match with the needed string. We assume that the input will be provided as a single file where the grammar is defined in a specific way and then the string being parsed and checked is inputted by the user.

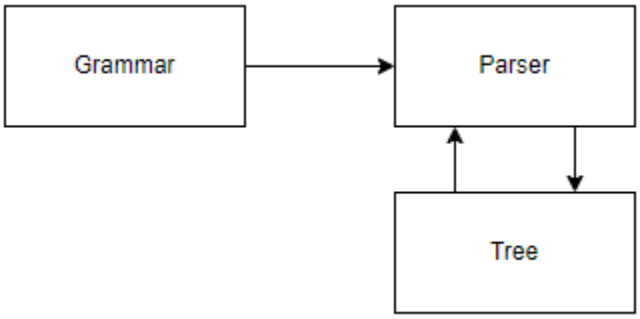
Functional requirements

The program will serve as a Top-Down parser with backtracking for the given grammar and input string. Based on the results of parsing the string through the grammar it will either accept it or throw an error if the input string as a character outside the defined terminal symbols. The steps of parsing the string will be output in text form. Only one .txt file will pass as an argument, syntax of the production of the grammar will follow a specific convention and the string being parsed will be inputted by the user also following a convention.

Implementation

General architecture

Below we see the structure of the classes which will be used.



The Grammar class takes in the input grammar provided by the user, interprets it, validates it to make sure it fits the proper criteria for grammar definition convention and then inserts the data into the appropriate data structures. To enable us to use any string as a terminal or non-terminal (e.g. 2 is terminal or 'bacon' is a non-terminal), we will assign each of the defined non-terminals as uppercase letters going in alphabetical order and assign each of the terminals as lowercase letters. It also validates

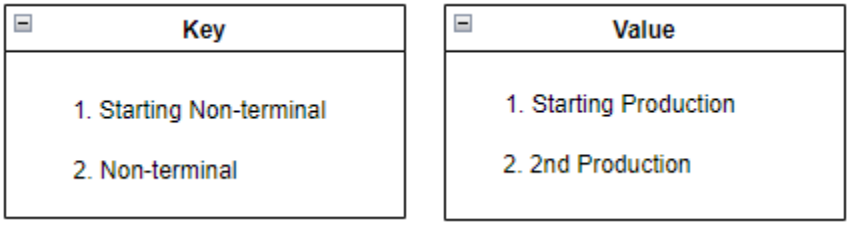
the input string and will throw an error if the input string contains characters outside the defined terminal symbols.

The *Parser* class is responsible for parsing the string through the production recursively and backtracking when the input string does not match the production. It utilizes checking the production for uppercase letters as non-terminals and lowercase letters as terminals and accordingly printing out the corresponding original definitions. This class will also print out the steps of the recursion for the user to see and print the final parse tree of the accepted string.

The *Tree* class stores parse tree nodes, keeping the symbol values (terminal, non-terminal and children). If the symbol is terminal, it is a leaf so the children array must be empty.

Data structures

The data structures used in this project will be a dictionary for storing the grammar. Within that dictionary there will be arrays used for each production. A tree will be created which will store parse tree nodes, this tree will be used to parse the input string through it recursively with the implementation of backtracking. Below is an example of the dictionary which will store the grammar, first data will have key for the starting non-terminal symbol, and value will be its production. Further we will store non-terminal symbols as keys and their production as values (which will be a combination of non-terminal and terminal symbols).



Class descriptions

Grammar Class

This class is responsible for interpreting, validating, and creating the grammar from the input file.

`def get_grammar_file():` Responsible for letting the user choose the input file with the grammar.

`def parse_file_to_grammar(file_path):` Responsible for using the path from the previous function, opening the file, interpreting the given input, validating them using functions like given below and creating the dictionary for the grammar.

`def validate_production_rule(rule, non_terminal):` Responsible for validating the input data, checking if the production has a non-terminal defined

`def get_input_string():` Responsible for asking the user for the string they want to parse and returning it to be used when parsing through the tree

In conclusion the *Grammar* class will create a dictionary for the grammar, with given productions if they are valid, and check if the input string is valid and return it for it to be used for the recursive algorithm.

Parser Class

This class does the main recursive traversal of the tree, where it takes the input string and the grammar produced from the *Grammar* class, then starts the traversal of the string through the given grammar. It expands the tree until it reaches the final leaf and then tries to match the string to the grammar, if it matches it moves onto the next part of the string, if it does not match it backtracks to the previous level of the tree and continues the recursion from there.

`def parser(grammar, input):` Responsible for starting the main recursion, calls recursive function from within it until the input string is parsed through the entire tree created by the *Tree* class until the tree is empty.

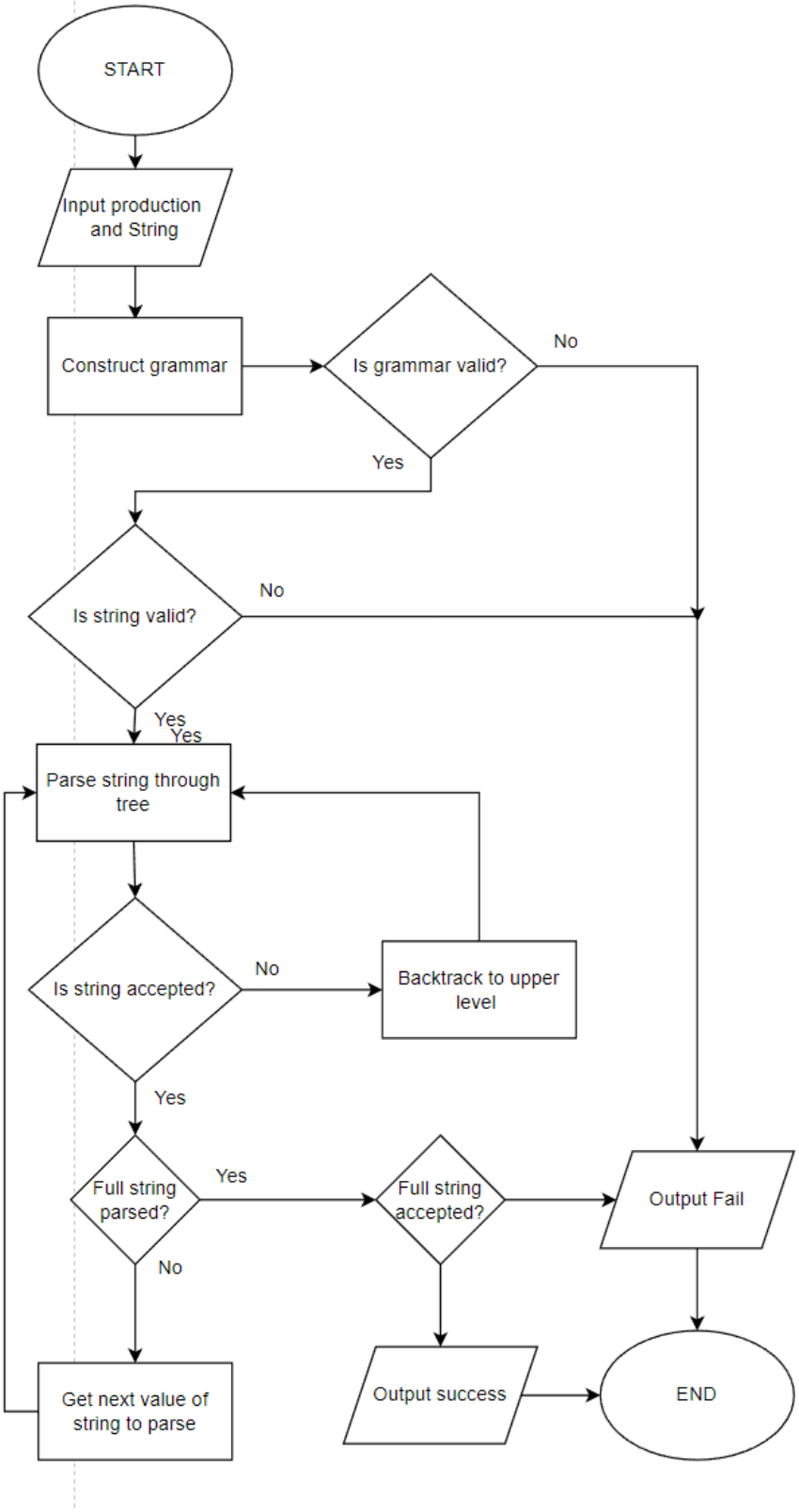
`def rec_parse(grammar, nonterminal, input, input_iter, out, node, parse_tree, matched):` Responsible for the recursion algorithm execution, creates a tree for the parsing, parses the input string through it, checks if the string matches the production, backtracks if it does not match and finally returns whether the string was accepted while printing every step of the recursion and the final parse tree.

Tree Class

This class is there to store the nodes of the tree which the input string will be parsed through.

```
class Tree:

    def __init__(self, symbol):

        self.symbol = symbol

        self.children = []

    def add_children(self, node):

        self.children.append(node)

    def remove_children(self):

        self.children = []

    def print_node(self):

        print("symbol: ", self.symbol)

        print("children: ", self.children)
```

General flow of the algorithm



Input/output description

The input will be a file containing the definition for the grammar. This file will be selected by the user as a prompt after the user executes the program using *python3 main.py*.

Rules:

1. The first line of the .txt file must describe the starting symbol using the format: #S ::= **insert starting symbol**
2. The next two lines of the .txt file must describe all non-terminal(excludes starting symbol) and terminal symbols, non-terminals are described using the format: #N ::= **insert non-terminal symbols/words/numbers**
3. The non-terminal symbols MUST be separated using |, e.g. #N ::= A|1|door|B
4. All terminal symbols are described using the format: #T ::= **insert terminal symbols/words/numbers**
5. The terminal symbols MUST be separated using |, e.g. #T ::= a|b|3|book|,
6. If non-terminal/terminal symbols are single letters, they MUST BE defined in ALPHABETICAL order
7. The following lines need to contain the productions, it must first start with the starting symbol and then followed by the production in the format **insert starting symbol* ::= *insert symbols** | **insert symbols** | **insert symbols**
8. The symbols inserted in the productions must be the ones defined in the second and third line of the file
9. Empty production can be defined using $ symbol, e.g. S::= a|$
10. If the file is in valid format, the program will prompt the user to input the string that they want to parse. This string must end with $, e.g. abcd$
11. Empty string can be denoted simply as $
12. Input file must be in .txt form

The output will be in the command line which will show the traversal of the string at each recursion of the algorithm, whether the string is finally accepted by that grammar and the production tree numbers which the parsed string went through. If an invalid string is inputted (meaning characters outside the defined terminal symbols are present in the string), the program will throw an exception before trying to parse the string through the tree. If there are any other errors in definition or production, then the program will also throw an exception.

Functional test cases

Error cases

Test 1: Non-terminal symbol not defined and used in production

```
#S ::= S
#N ::= A
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol A. Production rules must start with non-terminal symbols.
```

Test 11: Terminal symbol used as non-terminal

```
#S ::= S
#N ::= A
#T ::= a|b|c|d
S ::= aA|$
a ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol a. Production rules must start with non-terminal symbols.
```

Test 3: Incorrect syntax for production, missing separator

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S aA|$
A ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol S aA|$. Production rules must start with non-terminal symbols.
```

Test 4: Left recursion

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::= abA|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol A. Left recursion is not supported!
```

Test 5: Empty production

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::=
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol A. Production rule can not be empty!
```

Test 6: More than one production using the same non-terminal

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
A ::= $|d
```

```
SyntaxError: Non-terminal symbol A already exists in CFG!
```

Test 7: Terminal symbols not defined and used in production

```
#S ::= S
#N ::= A|B
#T ::= a|b
S ::= aA|$
A ::= Bc
B ::= $|d
```

```
SyntaxError: Invalid terminal symbols
```

Test 8: Change order of symbol definition

```
#T ::= a|b|c|d
#S ::= S
#N ::= A|B
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/test8.txt.txt
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}
Enter input string:
```

Allows program to run as normal

Test 9: Terminal symbols not defined and used in production

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB|abcde
B ::= $|d
```

```
SyntaxError: Invalid terminal symbols
```

Test 10: Invalid input string not ending in $

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/test10.txt.txt
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}
Enter input string: acd
```

```
SyntaxError: Invalid input string
```

Test 11: Invalid syntax for declaration of starting symbol

```
S ::= S
#N ::= A|B|S
#T ::= a|b|c|d
S ::= aA|$
B ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol S. Production rules must start with non-terminal symbols.
```

Test 12: Invalid syntax of non-terminal declaration

```
#S ::= S
N ::= A
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol N. Production rules must start with non-terminal symbols.
```

Test 13: Invalid syntax of terminal declaration

```
#S ::= S
#N ::= A
T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
SyntaxError: Error in non-terminal symbol T. Production rules must start with non-terminal symbols.
```

Test 14: Invalid position of terminal declaration

```
#S ::= S
#N ::= A
S ::= aA|$
A ::= abB|cB
B ::= $|d
#T ::= a|b|c|d
```

```
SyntaxError: Invalid terminal symbols
```

Test 15: Invalid input string

```
#S ::= S
#N ::= A|B
#T ::= a|b|c|d
S ::= aA|$
A ::= abB|cB
B ::= $|d
```

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/testmain.txt
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}
Enter input string: ace$
```

```
SyntaxError: Input string contains character: e outside the terminal symbols
```

Test 16: Invalid input file format, not in .txt

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/input_handler.py
```

```
NameError: Grammar file must be in .txt format.
```

Test 17: Blank input string

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/test10.txt.txt
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}
Enter input string:
```

```
SyntaxError: Invalid input string
```

Correct cases

Successful parsing

Test 1: Successful parsing of a string for given grammar

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/testmain.txt
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}
Enter input string: acd$
acd$

Provided contex-free grammar:
{'S': ['aA', '$'], 'A': ['abB', 'cB'], 'B': ['$', 'd']}

Provided input string: acd$
-----------------------------------
CFG: S

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a']
------------------------------------------------------------

Current production rule S : aA
Current input character: a

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a', 'A']
------------------------------------------------------------

Current production rule S : aA
Current input character: c
-----------------------------------
CFG: A

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['a']
------------------------------------------------------------

Current production rule A : abB
Current input character: c
               Backtracking...

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['c']
------------------------------------------------------------

Current production rule A : cB
Current input character: c

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['c', 'B']
------------------------------------------------------------

Current production rule A : cB
Current input character: d
-----------------------------------
CFG: B

-------------------------    PARSE TREE      --------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['c', 'B']
symbol:  B
children:  ['$']
------------------------------------------------------------

Current production rule B : $
Current input character: d
```

```
-------------------------    PARSE TREE      ----------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['c', 'B']
symbol:  B
children:  ['$', 'd']
-----------------------------------------------------------------

Current production rule B : d
Current input character: d


End of input parsing
Input string accepted by the CFG


-------------------------    PARSE TREE      ----------------
symbol:  S
children:  ['a', 'A']
symbol:  A
children:  ['c', 'B']
symbol:  B
children:  ['$', 'd']
-----------------------------------------------------------------
```

Test 2: Successful parsing of a string for given grammar

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/testbad.txt
{'S': ['doorala23,k[]alacool', '$'], 'door': ['ala23,', 'coolk[]'], ',': ['$', 'ala']}
Enter input string: coolk[]ala23alak[]alacool$

Provided contex-free grammar:
{'S': ['doorala23,k[]alacool', '$'], 'door': ['ala23,', 'coolk[]'], ',': ['$', 'ala']}

Provided input string: coolk[]ala23alak[]alacool$
----------------------------------
CFG: S

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door']
-----------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: cool
----------------------------------
CFG: door

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door']
symbol:  door
children:  ['ala']
-----------------------------------------------------------------------

Current production rule door : ala23,
Current input character: cool
              Backtracking...

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door']
symbol:  door
children:  ['cool']
-----------------------------------------------------------------------

Current production rule door : coolk[]
Current input character: cool

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door']
symbol:  door
children:  ['cool', 'k[]']
```

```
Current production rule door : coolk[]
Current input character: k[]

------------------------    PARSE TREE      --------------------
symbol:  S
children:  ['door', 'ala']
symbol:  door
children:  ['cool', 'k[]']
------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: ala

------------------------    PARSE TREE      --------------------
symbol:  S
children:  ['door', 'ala', '23']
symbol:  door
children:  ['cool', 'k[]']
------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: 23

------------------------    PARSE TREE      --------------------
symbol:  S
children:  ['door', 'ala', '23', ',']
symbol:  door
children:  ['cool', 'k[]']
------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: ala
---------------------------------
CFG: ,

------------------------    PARSE TREE      --------------------
symbol:  S
children:  ['door', 'ala', '23', ',']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$']
------------------------------------------------------------------

Current production rule , : $
Current input character: ala

------------------------    PARSE TREE      --------------------
symbol:  S
children:  ['door', 'ala', '23', ',']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$', 'ala']
------------------------------------------------------------------
```

```
Current production rule , : ala
Current input character: ala

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door', 'ala', '23', ',', 'k[]']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$', 'ala']
--------------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: k[]

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door', 'ala', '23', ',', 'k[]', 'ala']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$', 'ala']
--------------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: ala

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door', 'ala', '23', ',', 'k[]', 'ala', 'cool']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$', 'ala']
--------------------------------------------------------------------------

Current production rule S : doorala23,k[]alacool
Current input character: cool


End of input parsing
Input string accepted by the CFG


-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['door', 'ala', '23', ',', 'k[]', 'ala', 'cool']
symbol:  door
children:  ['cool', 'k[]']
symbol:  ,
children:  ['$', 'ala']
--------------------------------------------------------------------------
```

Test 3: Successful parsing of a string for given grammar

```
#S ::= S
#N ::= A|B|D|G|H
#T ::= a|b|c|d|e
S ::= A|Bce
A ::= b|D
D ::= c|B
B ::= d|aG
G ::= b|ed
H ::= e
```

```
PS C:\Users\AB\Desktop\ECOTE_Lab> python3 main.py
Provided file: C:/Users/AB/Desktop/ECOTE_Lab/test6.txt
{'S': ['A', 'Bce'], 'A': ['b', 'D'], 'D': ['c', 'B'], 'B': ['d', 'aG'], 'G': ['b', 'ed'], 'H': ['e']}
Enter input string: aed$

Provided contex-free grammar:
{'S': ['A', 'Bce'], 'A': ['b', 'D'], 'D': ['c', 'B'], 'B': ['d', 'aG'], 'G': ['b', 'ed'], 'H': ['e']}

Provided input string: aed$
----------------------------------
CFG: S

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
------------------------------------------------------------------------

Current production rule S : A
Current input character: a
----------------------------------
CFG: A

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['b']
------------------------------------------------------------------------

Current production rule A : b
Current input character: a
               Backtracking...

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
------------------------------------------------------------------------

Current production rule A : D
Current input character: a
----------------------------------
CFG: D

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['c']
------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------

Current production rule D : c
Current input character: a
               Backtracking...

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
--------------------------------------------------------------------------

Current production rule D : B
Current input character: a
-----------------------------------
CFG: B

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['d']
--------------------------------------------------------------------------

Current production rule B : d
Current input character: a
               Backtracking...

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a']
--------------------------------------------------------------------------

Current production rule B : aG
Current input character: a

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a', 'G']
--------------------------------------------------------------------------

Current production rule B : aG
Current input character: e
-----------------------------------
CFG: G
```

```
-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a', 'G']
symbol:  G
children:  ['b']
--------------------------------------------------------------------------

Current production rule G : b
Current input character: e
               Backtracking...

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a', 'G']
symbol:  G
children:  ['e']
--------------------------------------------------------------------------

Current production rule G : ed
Current input character: e

-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a', 'G']
symbol:  G
children:  ['e', 'd']
--------------------------------------------------------------------------

Current production rule G : ed
Current input character: d


End of input parsing
Input string accepted by the CFG


-------------------------    PARSE TREE      -------------------------
symbol:  S
children:  ['A']
symbol:  A
children:  ['D']
symbol:  D
children:  ['B']
symbol:  B
children:  ['a', 'G']
symbol:  G
children:  ['e', 'd']
--------------------------------------------------------------------------
```