The program consists of 3 dictionaries representing a valid input for its current state matched with the next token in the buffer.

We used a graph algorithm as we would shift or reduce to the next state based on the current state (top of the stack) and next token (front of queue) checking the adjacency list's until there is not valid move in the LR Parsing table.

We have 3 adjacency lists that checks for the next input in the buffer, the fist one is named **SHIFT**. This represents a valid shift action with the current input token and this would lead us to push the values [next_token_top_of_queue, next_state_from_shift]

```
    # {acepted_input: next state}
    shift = {
    '0id': '5',
    '0(': '4',
    '1+': '6',
    '2*': '7',
    '4id': '5',
    '4(': '4',
    '6id': '5',
    '6(': '4',
    '7id': '5',
    '7(': '4',
    '8+': '6',
    '8)': '11',
    '9*': '7'
}
```

The next adjacency list's that we are checking if the input is valid is named **REDUCE and GO_TO**. The key of the map represents the valid inputs, and the value represents a array of size 3 that includes [reduce_num: char, variable: char, pop_count: int]. If the next input represents a reduce action from our map, we can:
1. Pop from the stack pop_count times,
2. Check the replacement variable and check the next input is valid which is the top of the stack with replacement varible in our **GO_TO** map,
3. If it is valid, we push the replacement variable onto the stack and our next state from **GO_TO**.

```
# { state+input: [reduce_num: char, variable: char, pop_count: int] }
    reduce = {
    '2+': ['2', 'E', 2],
    '2)': ['2', 'E', 2],
    '2$': ['2', 'E', 2],
    '3+': ['4', 'T', 2],
    '3)': ['4', 'T', 2],
    '3$': ['4', 'T', 2],
    '3*': ['4', 'T', 2],
    '5+': ['6', 'F', 2],
    '5*': ['6', 'F', 2],
    '5)': ['6', 'F', 2],
    '5$': ['6', 'F', 2],
    '9+': ['1', 'E', 6],
    '9)': ['1', 'E', 6],
    '9$': ['1', 'E', 6],
    '10+': ['3', 'T', 6],
    '10)': ['3', 'T', 6],
    '10$': ['3', 'T', 6],
    '11+': ['5', 'F', 6],
    '11)': ['5', 'F', 6],
    '11*': ['5', 'F', 6],
    '11$': ['5', 'F', 6]
}

# {state+input: next_state}
    go_to = {
    '0E' : '1',
    '0T' : '2',
    '0F' : '3',
    '4E' : '8',
    '4T' : '2',
    '4F' : '3',
    '6T' : '9',
    '6F' : '3',
    '7F' : '10',
}
```

There is a while loop that runs until the current input is not valid in either **REDUCE** or **SHIFT** then we can check if the input is valid by comparing the top of the stack with state 1 and front of the queue as $.