Lexical analyzer

compile_patterns

- **Time Complexity**: **O(n)**, where n is the number of patterns in token_patterns.

analyze

- **Time Complexity**: **O(m * n)**, where m is the length of the input line and n is the number of patterns in compiled_patterns.

missing_semicolon

- **Time Complexity**: **O(m)**, where m is the length of the input line.

detect_wrong_allocations

- **Time Complexity**: **O(m)**, where m is the length of the input line.

manage_tokens

- **Time Complexity**: **O(k * (m * n + m))**, where:
  - k is the number of lines in the input code.
  - m is the average length of a line.
  - n is the number of patterns in compiled_patterns.

None recursive predictive parser

parse

- **Time Complexity**: **O(n * m)**, where:
  - n is the number of input tokens.
  - m is the maximum number of productions applied for a single token.

Parse table

add_production

- **Time Complexity**: **O(1)** (amortized)
    - Adds a production to the rules dictionary. Appending to a list is **O(1)** on average.

---

fill

- **Time Complexity**: **O(1)**
    - Adds a fixed number of productions to the grammar. The number of productions is constant, so this method is **O(1)**.

---

calculate_first

- **Time Complexity**: **O(V * P * S)**, where:
    - V is the number of non-terminals (variables).
    - P is the maximum number of productions for a non-terminal.
    - S is the maximum length of a production.
- **Explanation**:
    - The outer while loop runs until no changes occur in the first sets. In the worst case, this can take **O(V)** iterations.
    - The inner loops iterate over all productions and symbols in each production, leading to **O(P * S)** complexity per iteration.
    - Overall, the complexity is **O(V * P * S)**.

---

calculate_follow

- **Time Complexity**: **O(V * P * S)**, where:
    - V is the number of non-terminals.
    - P is the maximum number of productions for a non-terminal.
    - S is the maximum length of a production.

---

first_of_sequence

- **Time Complexity**: **O(S)**, where S is the length of the input sequence.

construct_parse_table

- **Time Complexity**: **O(V * T * P * S)**, where:
    - V is the number of non-terminals.
    - T is the number of terminals.
    - P is the maximum number of productions for a non-terminal.
    - S is the maximum length of a production.

---

save_to_file

- **Time Complexity**: **O(V * T)**, where:
    - V is the number of non-terminals.
    - T is the number of terminals.

Parse tree

add_production

- **Time Complexity: O(1)**

fill

- **Time Complexity: O(1)**

calculate_first

- **Time Complexity: O(V * P * S),** where:
    - V is the number of non-terminals (variables).
    - P is the maximum number of productions for a non-terminal.
    - S is the maximum length of a production.

calculate_follow

- **Time Complexity: O(V * P * S),** where:
    - V is the number of non-terminals.
    - P is the maximum number of productions for a non-terminal.
    - S is the maximum length of a production.

first_of_sequence

- **Time Complexity: O(S),** where S is the length of the input sequence.

construct_parse_table

- **Time Complexity: O(V * T * P * S),** where:
  - V is the number of non-terminals.
  - T is the number of terminals.
  - P is the maximum number of productions for a non-terminal.
  - S is the maximum length of a production.

save_to_file

- **Time Complexity: O(V * T),** where:
  - V is the number of non-terminals.
  - T is the number of terminals.

Parse table

hash_function

- **Time Complexity: O(n),** where n is the length of the key.

create_table

- **Time Complexity: O(m * k),** where:
  - m is the number of elements in elements.
  - k is the maximum number of tokens in any category (self.tokens[element]).

manage_tokens

- **Time Complexity: O(t * (n + k log k)),** where:
  - t is the number of tokens in token_list.
  - n is the average length of a token.
  - k is the maximum number of tokens in any category (self.tokens[key])

__init__

- **Time Complexity: O(t * (n + k log k) + m * k)**, where:
    - t is the number of tokens in token_list.
    - n is the average length of a token.
    - k is the maximum number of tokens in any category (self.tokens[key]).
    - m is the number of elements in elements.

Search In Tree

find_declaration

- **Time Complexity: O(V + E),** where:
    - V is the number of nodes (vertices) in the tree.
    - E is the number of edges in the tree.

process_l_node

- **Time Complexity: O(V + E),** where:
    - V is the number of nodes in the subtree rooted at l_node.
    - E is the number of edges in the subtree rooted at l_node.

find_terminal

- **Time Complexity: O(h)**, where h is the height of the subtree rooted at node.

find_number_node

- **Time Complexity: O(V + E),** where:
    - V is the number of nodes in the subtree rooted at node.
    - E is the number of edges in the subtree rooted at node.