```
class RecursiveDescentParser:
    def __init__(self, grammar):
        self.grammar = grammar
        self.current_token = None
        self.index = 0
        self.input_string = []
        self.parse_tree = []
        self.node_counter = 0

    def parse(self, input_string):
        self.input_string = input_string.split()
        self.index = 0
        self.parse_tree = []

        # Start the parsing process with the start symbol
        success = self.parse_nonterminal(self.grammar.start_symbol, None)

        if success and self.index == len(self.input_string):
            print("Parsing successful.")
            self.print_parse_tree()
        else:
            print("Parsing failed.")

    def parse_nonterminal(self, nonterminal, parent):
        # Store parent-child relationship in parse tree
        node_id = self.node_counter
        self.node_counter += 1
        self.parse_tree.append((node_id, nonterminal, parent))

        # Try all productions for this nonterminal
        for production in self.grammar.productions[nonterminal]:
            saved_index = self.index

            if all(self.parse_symbol(symbol, node_id) for symbol in production):
                return True

            self.index = saved_index

        return False

    def parse_symbol(self, symbol, parent):
        if symbol in self.grammar.nonterminals:
            return self.parse_nonterminal(symbol, parent)
        elif self.index < len(self.input_string) and symbol == self.input_string[self.index]:
            # Match terminal
            node_id = self.node_counter
            self.node_counter += 1
            self.parse_tree.append((node_id, symbol, parent))
            self.index += 1
            return True

        return False

    def print_parse_tree(self):
        print("Parse Tree (ID, Symbol, Parent):")
```

```
for node_id, symbol, parent in self.parse_tree:
    print(f"Node ID: {node_id}, Symbol: {symbol}, Parent ID: {parent}")
```