تحلیل الگوریتم cyk

محمد مهدی حیدری
۹۴۳۱۳۰۶

۱۶ خرداد ۱۳۹۸

در فایل cyk_main.py چند تابع کمکی برای خواندن ورودی و گرامر از فایل متنی نوشته شده. در انتهای هر فایل باید یک خط خالی وجود داشته باشد. سمت راست و چپ هر قانون گرامر باید با $\rightarrow$ جدا شده باشد و نتایج آن می‌تواند با |، or شود. یک تابع نیز برای نمایش نتیجه الگوریتم نوشته شده که در ابتدای هر خط طول زیرمجموعه‌های مورد بررسی از ورودی نوشته شده است. نتیجه‌ی موجود در هر خانه با { } مشخص شده و اگر خالی باشد به جای آن ـ گذاشته شده. در تابع مربوط به الگوریتم ابتدا ترمینال‌های موجود در ورودی در نظر گرفته می‌شوند. قواعدی که منجر به تولید ترمینال می‌شوند به طور جداگانه ذخیره شده‌اند و فقط در مرحله اول مورد نیاز هستند. برای هر ترمینال در ورودی متغیری در را پیدا می‌کنیم که در سمت چپ یک قاعده که آن ترمینال را تولید می‌کند، حضور داشته باشد. در مرحله بعد زیر رشته‌های دوتایی را بررسی می‌کنیم. برای رخ دادن هرکدام باید در سمت راست قواعدی که دو متغیر تولید می‌کنند به دنبال آنها بگردیم و متغیر سمت چپ را ذخیره کنیم. در مراحل بعدی برای هر زیررشته ورودی چک می‌کنیم که تحت چه حالت‌هایی قابل ساخته شدن هستند. مثلا aba از ترکیب a + ab یا ba + a تولید می‌شود که نتایج هر یک در جدول موجود است و با یک تابع کمکی تمام حالت‌های ممکن را می‌سازیم و باز به دنبال حالت‌های رخداد آن می‌گردیم در مرحله آخر اگر متغیر آغازی در نتیجه وجود داشته باشد یعنی می‌توانیم با شروع از آن و استفاده از قواعد موجود در گرامر، رشته داده شده را بسازیم.

```python
import os


def create_cell(first, second):
    """
    creates set of string from concatenation of each character in first
    to each character in second
    :param first: first set of characters
    :param second: second set of characters
    :return: set of desired values
    """
    res = set()
    if first == set() or second == set():
        return set()
    for f in first:
        for s in second:
            res.add(f+s)
    return res


def read_grammar(filename="./grammar.txt"):
    """
    reads the rules of a context free grammar from a text file
    :param filename: name of the text file in current directory
    :return: two lists. v_rules lead to variables and t_rules
    lead to terminals.
    """
    filename = os.path.join(os.curdir, filename)
    with open(filename) as grammar:
        rules = grammar.readlines()
        v_rules = []
        t_rules = []

        for rule in rules:
            left, right = rule.split(" -> ")

            # for two or more results from a variable
            right = right[:-1].split(" | ")
            for ri in right:

                # it is a terminal
                if str.islower(ri):
                    t_rules.append([left, ri])

                # it is a variable
                else:
                    v_rules.append([left, ri])
        return v_rules, t_rules


def read_input(filename="./input.txt"):
    """
    reads the inputs from a text file
    :param filename: name of the text file in current directory
    :return: list of inputs
    """
    filename = os.path.join(os.curdir, filename)
    res = []
    with open(filename) as inp:
        inputs = inp.readlines()
        for i in inputs:

            # remove \n
            res.append(i[:-1])
    return res


def cyk_alg(varies, terms, inp):
    """
    implementation of CYK algorithm
    :param varies: rules related to variables
    :param terms: rules related to terminals
    :param inp: input string
```

```
    :return: resulting table
    """

    length = len(inp)
    var0 = [va[0] for va in varies]
    var1 = [va[1] for va in varies]

    # table on which we run the algorithm
    table = [[set() for _ in range(length-i)] for i in range(length)]

    # Deal with variables
    for i in range(length):
        for te in terms:
            if inp[i] == te[1]:
                table[0][i].add(te[0])

    # Deal with terminals
    # its complexity is O(|G|*n^3)
    for i in range(1, length):
        for j in range(length - i):
            for k in range(i):
                row = create_cell(table[k][j], table[i-k-1][j+k+1])
                for ro in row:
                    if ro in var1:
                        table[i][j].add(var0[var1.index(ro)])

    # if the last element of table contains "S" the input belongs to the grammar
    return table


def show_result(tab, inp):
    """
    this function prints the procedure of cyk.
    in the end there is a message showing if the input
    belongs to the grammar
    :param tab: table
    :param inp: input
    :return: None
    """
    for c in inp:
        print("\t{}".format(c), end="\t")
    print()
    for i in range(len(inp)):
        print(i+1, end="")
        for c in tab[i]:
            if c == set():
                print("\t{}".format("_"), end="\t")
            else:
                print("\t{}".format(c), end=" ")
        print()

    if 'S' in tab[len(inp)-1][0]:
        print("The input belongs to this context free grammar!")
    else:
        print("The input does not belong to this context free grammar!")


if __name__ == '__main__':
    v, t = read_grammar()
    r = read_input()[0]
    ta = cyk_alg(v, t, r)
    show_result(ta, r)
```

برای مشاهده کد می‌توانید به اینجا مراجعه کنید.