《编译原理》专题4设计

目标任务

实验项目

实现算符优先分析算法，完成以下描述算术表达式的算符优先文法的算符优先分析过程。

```
G[E]:E→E+T∣E-T∣T
T→T*F∣T/F∣F
F→(E)∣i
```

设计说明

终结符号i为用户定义的简单变量,即标识符的定义。

设计要求

1. 构造该算符优先文法的优先关系矩阵或优先函数；
2. 输入串应是词法分析的输出二元式序列，即某算术表达式“专题1”的输出结果。输出为输入串是否为该文法定义的算术表达式的判断结果。
3. 算符优先分析过程应能发现输入串出错。
4. 设计两个测试用例（尽可能完备，正确和出错），并给出测试结果；（4）考虑根据算符优先文法构造算符优先关系矩阵，包括FIRSTVT和LASTVT集合，并添加到你的算符优先分析程序中。

程序功能描述

1. 对二元式算符优先识别和分析
2. 根据输入的二元式内容进行分析语法分析,输出结果
3. 输出分析过程分析栈和保留串和错误提示
4. 构造FIRSTVT集和LASTVT集
5. 构造出算符优先矩阵分析表
6. 构造算符优先矩阵分析器

数据结构

存储规则左部和右部字符的类

```python
class Rule(object):
    def __init__(self):
        self.left = ""
        self.right = []
```

文法和分析器

```python
#终结符
VT = []
#非终结符
VN = []
#规则集合
Rules = []
# FirstVT集与LastVT集
FirstVT = []
LastVT = []
#存储规则左部和右部的列表
rule_list = []
#优先关系矩阵
OG = []
#分析栈
og_stack = []
```

程序结构描述

算符优先分析法分析过程

```python
def og():
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readlines()
    for i in range(len(src)):
        flag = False
        og_stack = []
        src[i] = src[i].replace('\n', '')
        current = 0
        pos = 1
        og_stack.append('#')
        while current != len(src[i]):
            a = src[i][current]
            s = og_stack[pos-1]
            if s in VT:
                j = pos
            else:
                j = pos - 1
            while pos != 2 or a != '#':
                if OG[VT.index(s)][VT.index(a)] == '>':
                    while True:
                        q = s
                        j = j - 1
                        s = og_stack[j-1]
                        if s not in VT:
                            j = j - 1
                            s = og_stack[j-1]
                        if OG[VT.index(s)][VT.index(q)] == '<':
                            pos = j + 1
                            if pos == len(og_stack):
                                og_stack[pos-1] = 'N'
                            else:
                                while pos != len(og_stack):
                                    og_stack.pop()
                            break
                else:
                    og_stack.append(a)
                    pos = pos + 1
                    current = current + 1
                    break
            if pos == 2 and a == '#':
                flag = True
                break
        with open('output.txt', 'a', encoding='utf-8') as out_file:
            if flag:
                out_file.write('%s为合法字符串\n' % src[i])
            else:
                out_file.write('%s为不合法字符串\n' % src[i])
}
```

打印算符优先分析表

```
def print_og():
    with open('OG.txt', 'w', encoding='utf-8') as chart_write:
        chart_write.write('生成的优先矩阵如下\n')
        for c in VT:
            chart_write.write("%s       \t" % c)
        chart_write.write("\n")
        for i in range(len(OG)):
            for j in range(len(OG[i])):
                chart_write.write("%s       \t" % OG[i][j])
            chart_write.write("\n")
```

得到每个规则的左部和右部

```python
def create_rule_list():
    for i in range(0, len(Rules)):
        # 去掉空格
        Rules[i] = Rules[i].replace(' ', '')
        rule = Rule()
        rule_list.append(rule)
    for j in range(0, len(Rules)):
        arrow_pos = Rules[j].find('-')
        rule_list[j].left = Rules[j][0:arrow_pos]
        #将规则右部转换成列表
        rule_list[j].right = list(Rules[j][arrow_pos + 2:])
        while "'" in rule_list[j].right:
            pos = rule_list[j].right.index("'")
            new_sym = "".join(rule_list[j].right[pos - 1: pos + 1])
            del rule_list[j].right[pos]
            del rule_list[j].right[pos - 1]
            if new_sym not in rule_list[j].right:
                rule_list[j].right.append(new_sym)
```

从输入的规则中找出终结符和非终结符

```python
def identify_vt_and_vn():
    for i in range(0, len(rule_list)):
        #把规则左部加入到非终结符集合中
        if rule_list[i].left not in VN:
            VN.append(rule_list[i].left)
        #将规则右部的终结符和非终结符加入到相应的集合
        for j in range(len(rule_list[i].right)):
            if rule_list[i].right[j].isupper():
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
            elif rule_list[i].right[j] != 'ε' and "'" not in
rule_list[i].right[j]:
                if rule_list[i].right[j] not in VT:
                    VT.append(rule_list[i].right[j])
            elif "'" in rule_list[i].right[j]:
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
    VT.append('#')
```

创建FirstVT集

```
def gen_firstvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            # 形如U -> b...之类的规则，将b加入U的FirstVT集
            if rule_list[i].right[0] in VT:
                if rule_list[i].right[0] not in FirstVT[VN.index(ch)]:
                    FirstVT[VN.index(ch)].append(rule_list[i].right[0])
            # 形如U -> Vb...之类的规则，将b加入U的FirstVT集
            elif len(rule_list[i].right) > 1 and rule_list[i].right[1] in
VT:
                if rule_list[i].right[1] not in FirstVT[VN.index(ch)]:
                    FirstVT[VN.index(ch)].append(rule_list[i].right[1])
            # 形如U -> V...的规则，将V的FirstVT集里的元素加入U的FirstVT集
            if rule_list[i].right[0] in VN:
                if not FirstVT[VN.index(rule_list[i].right[0])]:
                    gen_firstvt(rule_list[i].right[0])
                for c in FirstVT[VN.index(rule_list[i].right[0])]:
                    if c not in FirstVT[VN.index(ch)]:
                        FirstVT[VN.index(ch)].append(c)
```

创建LastVT集

```
def gen_lastvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            # 形如U -> ...a之类的规则，将a加入U的LastVT集
            if rule_list[i].right[-1] in VT:
                if rule_list[i].right[-1] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-1])
            # 形如U -> ...aV之类的规则，将a加入U的LastVT集
            elif len(rule_list[i].right) > 1 and rule_list[i].right[-2] in
VT and rule_list[i].right[-1] in VN:
                if rule_list[i].right[-2] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-2])
            # 形如U -> ...V的规则，将V的LastVT集里的元素加入U的LastVT集
            if rule_list[i].right[-1] in VN:
                if not LastVT[VN.index(rule_list[i].right[-1])]:
                    gen_lastvt(rule_list[i].right[-1])
                for c in LastVT[VN.index(rule_list[i].right[-1])]:
                    if c not in LastVT[VN.index(ch)]:
                        LastVT[VN.index(ch)].append(c)
```

创建优先关系矩阵

```python
def create_og():
    for i in range(len(rule_list)):
        for j in range(0, len(rule_list[i].right) - 1):
            # 形如U -> ...ab...的规则, 则a = b
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+1])] = '='
            # 形如U -> ...aVb...的规则, 则a = b
            if j < len(rule_list[i].right) - 2 and rule_list[i].right[j] in VT and rule_list[i].right[j+2] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+2])] = '='
            # 形如U -> ...aU...的规则, 则a < U的FirstVT集中的元素
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VN:
                for c in FirstVT[VN.index(rule_list[i].right[j+1])]:
                    OG[VT.index(rule_list[i].right[j])][VT.index(c)] = '<'
            # 形如U -> ...Ub...的规则, 则U的LastVT集中的元素 > b
            if rule_list[i].right[j] in VN and rule_list[i].right[j+1] in VT:
                for c in LastVT[VN.index(rule_list[i].right[j])]:
                    OG[VT.index(c)][VT.index(rule_list[i].right[j+1])] = '>'
    # # <起始符号的FirstVT集中的元素, 起始符号的LastVT集的元素 > #
    for c in FirstVT[VN.index(rule_list[0].left)]:
        OG[VT.index('#')][VT.index(c)] = '<'
    for c in LastVT[VN.index(rule_list[0].left)]:
        OG[VT.index(c)][VT.index('#')] = '>'
```

测试

测试用例输入

```
i+i*i#
i*(i+i)#
(i+i*i#
```

测试用例输出

```
i+i*i#为合法字符串
i*(i+i)#为合法字符串
(i+i*i#为不合法字符串
```

源代码

```python
#存储规则左部和右部字符的类
class Rule(object):
    def __init__(self):
        self.left = ""
        self.right = []


#终结符
VT = []
#非终结符
VN = []
#规则集合
Rules = []
# FirstVT集与LastVT集
FirstVT = []
LastVT = []
#存储规则左部和右部的列表
rule_list = []
#优先关系矩阵
OG = []
#分析栈
og_stack = []


# 得到每个规则的左部和右部
def create_rule_list():
    for i in range(0, len(Rules)):
        # 去掉空格
        Rules[i] = Rules[i].replace(' ', '')
        rule = Rule()
        rule_list.append(rule)
    for j in range(0, len(Rules)):
        arrow_pos = Rules[j].find('-')
        rule_list[j].left = Rules[j][0:arrow_pos]
        #将规则右部转换成列表
        rule_list[j].right = list(Rules[j][arrow_pos + 2:])
        while "'" in rule_list[j].right:
            pos = rule_list[j].right.index("'")
            new_sym = "".join(rule_list[j].right[pos - 1: pos + 1])
            del rule_list[j].right[pos]
            del rule_list[j].right[pos - 1]
            if new_sym not in rule_list[j].right:
                rule_list[j].right.append(new_sym)


# 从输入的规则中找出终结符和非终结符
```

```python
def identify_vt_and_vn():
    for i in range(0, len(rule_list)):
        #把规则左部加入到非终结符集合中
        if rule_list[i].left not in VN:
            VN.append(rule_list[i].left)
        #将规则右部的终结符和非终结符加入到相应的集合
        for j in range(len(rule_list[i].right)):
            if rule_list[i].right[j].isupper():
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
            elif rule_list[i].right[j] != 'ε' and "'" not in
rule_list[i].right[j]:
                if rule_list[i].right[j] not in VT:
                    VT.append(rule_list[i].right[j])
            elif "'" in rule_list[i].right[j]:
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
    VT.append('#')


# 创建FirstVT集
def gen_firstvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            # 形如U -> b...之类的规则，将b加入U的FirstVT集
            if rule_list[i].right[0] in VT:
                if rule_list[i].right[0] not in FirstVT[VN.index(ch)]:
                    FirstVT[VN.index(ch)].append(rule_list[i].right[0])
            # 形如U -> Vb...之类的规则，将b加入U的FirstVT集
            elif len(rule_list[i].right) > 1 and rule_list[i].right[1] in
VT:
                if rule_list[i].right[1] not in FirstVT[VN.index(ch)]:
                    FirstVT[VN.index(ch)].append(rule_list[i].right[1])
            # 形如U -> V...的规则，将V的FirstVT集里的元素加入U的FirstVT集
            if rule_list[i].right[0] in VN:
                if not FirstVT[VN.index(rule_list[i].right[0])]:
                    gen_firstvt(rule_list[i].right[0])
                for c in FirstVT[VN.index(rule_list[i].right[0])]:
                    if c not in FirstVT[VN.index(ch)]:
                        FirstVT[VN.index(ch)].append(c)


# 创建LastVT集
def gen_lastvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            # 形如U -> ...a之类的规则，将a加入U的LastVT集
            if rule_list[i].right[-1] in VT:
                if rule_list[i].right[-1] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-1])
            # 形如U -> ...aV之类的规则，将a加入U的LastVT集
            elif len(rule_list[i].right) > 1 and rule_list[i].right[-2] in VT and rule_list[i].right[-1] in VN:
                if rule_list[i].right[-2] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-2])
            # 形如U -> ...V的规则，将V的LastVT集里的元素加入U的LastVT集
            if rule_list[i].right[-1] in VN:
                if not LastVT[VN.index(rule_list[i].right[-1])]:
                    gen_lastvt(rule_list[i].right[-1])
                for c in LastVT[VN.index(rule_list[i].right[-1])]:
                    if c not in LastVT[VN.index(ch)]:
                        LastVT[VN.index(ch)].append(c)


# 创建优先关系矩阵
def create_og():
    for i in range(len(rule_list)):
        for j in range(0, len(rule_list[i].right) - 1):
            # 形如U -> ...ab...的规则，则a = b
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+1])] = '='
            # 形如U -> ...aVb...的规则，则a = b
            if j < len(rule_list[i].right) - 2 and rule_list[i].right[j] in VT and rule_list[i].right[j+2] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+2])] = '='
            # 形如U -> ...aU...的规则，则a < U的FirstVT集中的元素
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VN:
                for c in FirstVT[VN.index(rule_list[i].right[j+1])]:
                    OG[VT.index(rule_list[i].right[j])][VT.index(c)] = '<'
            # 形如U -> ...Ub...的规则，则U的LastVT集中的元素 > b
            if rule_list[i].right[j] in VN and rule_list[i].right[j+1] in VT:
                for c in LastVT[VN.index(rule_list[i].right[j])]:
                    OG[VT.index(c)][VT.index(rule_list[i].right[j+1])] = '>'
    # # <起始符号的FirstVT集中的元素，起始符号的LastVT集的元素 > #
    for c in FirstVT[VN.index(rule_list[0].left)]:
        OG[VT.index('#')][VT.index(c)] = '<'
    for c in LastVT[VN.index(rule_list[0].left)]:
        OG[VT.index(c)][VT.index('#')] = '>'


# 算符优先分析法分析过程
def og():
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readlines()
    for i in range(len(src)):
        flag = False
        og_stack = []
        src[i] = src[i].replace('\n', '')
        current = 0
        pos = 1
        og_stack.append('#')
        while current != len(src[i]):
            a = src[i][current]
            s = og_stack[pos-1]
            if s in VT:
                j = pos
            else:
                j = pos - 1
            while pos != 2 or a != '#':
                if OG[VT.index(s)][VT.index(a)] == '>':
                    while True:
                        q = s
                        j = j - 1
                        s = og_stack[j-1]
                        if s not in VT:
                            j = j - 1
                            s = og_stack[j-1]
                        if OG[VT.index(s)][VT.index(q)] == '<':
                            pos = j + 1
                            if pos == len(og_stack):
                                og_stack[pos-1] = 'N'
                            else:
                                while pos != len(og_stack):
                                    og_stack.pop()
                            break
                else:
                    og_stack.append(a)
                    pos = pos + 1
                    current = current + 1
                    break
            if pos == 2 and a == '#':
                flag = True
                break
        with open('output.txt', 'a', encoding='utf-8') as out_file:
            if flag:
                out_file.write('%s为合法字符串\n' % src[i])
            else:
                out_file.write('%s为不合法字符串\n' % src[i])


# 打印FirstVT集和LastVT集
def print_vt():
```

```python
    with open('set.txt', 'w', encoding='utf-8') as set_file:
        set_file.write("生成的FirstVT集如下\n")
        for k in range(len(VN)):
            set_file.write("%3s:\t" % VN[k])
            for p in FirstVT[k]:
                set_file.write("%s\t" % p)
            set_file.write("\n")
            set_file.write("\n")
        set_file.write("生成的LastVT集如下\n")
        for m in range(len(VN)):
            set_file.write("%3s:\t" % VN[m])
            for n in LastVT[m]:
                set_file.write("%s\t" % n)
            set_file.write("\n")
            set_file.write("\n")


# 打印算符优先分析表
def print_og():
    with open('OG.txt', 'w', encoding='utf-8') as chart_write:
        chart_write.write('生成的优先矩阵如下\n')
        for c in VT:
            chart_write.write("%s       \t" % c)
        chart_write.write("\n")
        for i in range(len(OG)):
            for j in range(len(OG[i])):
                chart_write.write("%s       \t" % OG[i][j])
            chart_write.write("\n")


if __name__ == '__main__':
    with open('rule.txt', 'r', encoding='utf-8') as rule_file:
        Rules = rule_file.readlines()
        for i in range(len(Rules)):
            Rules[i] = Rules[i].replace('\n', '')
    create_rule_list()
    identify_vt_and_vn()
    for j in range(len(VN)):
        FirstVT.append([])
        LastVT.append([])
    OG = [[0 for col in range(len(VT))]for row in range(len(VT))]
    for k in range(len(VN)):
        gen_firstvt(VN[k])
    for p in range(len(VN)):
        gen_lastvt(VN[p])
    print_vt()
    create_og()
    print_og()
    og()
```