第二次实验

实验目的

1. 理解算符优先文法的作用
2. 学会构造**FIRSTVT**和**LASTVT**，以及利用这两个集合构造优先关系矩阵
3. 理解算符优先文法移进与归约的过程
4. 编码实现一般算符优先文法的过程

实验步骤

1. 定义文法和句子输入的规则
2. 根据输入的文法得出文法的非终结符与终结符集合
3. 实现书上**FIRSTVT**和**LASTVT**的伪代码
4. 根据得到的**FIRSTVT**和**LASTVT**构造优先关系矩阵
5. 利用优先关系矩阵对输入句子进行分析
6. 完成输出格式与出错处理
7. 完善前端UI设计
8. 部署项目到服务器上

程序分析

程序访问：http://compile.lkc1621.xyz/opg

这里只给出算符优先分析器的代码进行分析，完整程序请看文件夹。

```
#! env python3
# -*- coding: UTF-8 -*-

from collections import namedtuple, OrderedDict
from random import choice
from exceptions import NotOPGError, OPGRunError

Rule = namedtuple('Rule', 'left, right')


class OPGEngine:
    def __init__(self):
        self.rules = []
        self.V_n = set()
        self.V_t = set()
        self.priority_table = dict()

    def get_rules(self, grammar):
        self.rules.clear()
        self.V_n.clear()
        self.V_t.clear()
        self.priority_table.clear()

        def get_random_vn():
```

```
            all_c = set(c for c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
            left_c = all_c - self.V_n
            return choice(list(left_c))

        t_V = set()
        for rule in grammar:
            left, right = rule.split('->')
            self.V_n.add(left)
            for item in right.split('|'):
                self.rules.append(Rule(left, item))
        for rule in self.rules:
            for item in rule.right:
                t_V.add(item)
        for v in t_V:
            if v not in self.V_n:
                self.V_t.add(v)

        # add a rule like "S->#E#"
        vn = get_random_vn()
        self.V_n.add(vn)
        # the first rule of the grammar must be the entry
        t_g = grammar[0].split('->')[0]
        self.rules.insert(0, Rule(vn, '#' + t_g + '#'))
        self.V_t.add('#')
        # print(self.V_n)
        # print(self.V_t)
        # print(self.rules)

    def get_priority_table(self):
        def cal_needed_vt(mode=0): # mode:0=>firstvt, mode:1=>lastvt
            stack = list()
            F = list()
            i0 = 0
            i1 = 1

            def insert_f(u, b):
                if (u, b) not in F:
                    F.append((u, b))
                    stack.append((u, b))

            if mode == 1:
                i0 = -1
                i1 = -2
            for rule in self.rules:
                if rule.right[i0] in self.V_t:
                    insert_f(rule.left, rule.right[i0])
                if len(rule.right) >=2 and rule.right[i0] in self.V_n and rule.right[i1] in
self.V_t:
                    insert_f(rule.left, rule.right[i1])
            while len(stack) > 0:
                v, b = stack.pop()
                for rule in self.rules:

                    if rule.right[i0] == v:
```

```
                        insert_f(rule.left, b)
            return F

        first_vt = cal_needed_vt(0)
        last_vt = cal_needed_vt(1)
        # print(first_vt)
        # print(last_vt)

        def insert(key, value):
            table = self.priority_table
            if key in table and table[key] != value:
                raise NotOPGError
            else:
                table[key] = value

        try:
            for rule in self.rules:
                right = rule.right
                length = len(right)
                i = 0
                while i < length - 1:
                    if right[i] in self.V_t and right[i+1] in self.V_t:
                        insert((right[i], right[i+1]), '=')
                    if i < length - 2 and right[i] in self.V_t \
                            and right[i+1] in self.V_n and right[i+2] in self.V_t:
                        insert((right[i], right[i+2]), '=')
                    if right[i] in self.V_t and right[i+1] in self.V_n:
                        for u, b in first_vt:
                            if u == right[i+1]:
                                insert((right[i], b), '<')
                    if right[i] in self.V_n and right[i+1] in self.V_t:
                        for u, b in last_vt:
                            if u == right[i]:
                                insert((b, right[i+1]), '>')
                    i += 1
            return True
        except NotOPGError as e:
            print(e.message)
            return False

    def print_priority_table(self):
        print('\t', end='')
        for vt2 in self.V_t:
            print('{0}\t'.format(vt2), end='')
        print()
        for vt1 in self.V_t:
            print('{0}\t'.format(vt1), end='')
            for vt2 in self.V_t:
                priority = self.priority_table.get((vt1, vt2))
                if not priority:
                    priority = '?'
                print('{0}\t'.format(priority), end='')

            print()
```

```
    def complete_priority_table(self):
        res = []
        if not self.get_priority_table():
            res.append({
                'state': 'error',
                'message': 'It isn\'t OPG'
            })
        else:
            res.append({
                'state': 'normal',
                'length': len(self.V_t) + 1
            })
            t1 = OrderedDict()
            t1['c1'] = ''
            i = 1
            for vt2 in self.V_t:
                i += 1
                t1['c'+str(i)] = vt2
            res.append(t1)
            for vt1 in self.V_t:
                t2 = OrderedDict()
                t2['c1'] = vt1
                i = 1
                for vt2 in self.V_t:
                    i += 1
                    priority = self.priority_table.get((vt1, vt2))
                    if not priority:
                        priority = '?'
                    t2['c' + str(i)] = priority
                res.append(t2)
        return res

    def analyse(self, sentence):
        stack = list()
        stack.append('#')
        sentence = sentence + '#'
        cur = 0
        step = 0
        length = len(sentence)

        def reduce(part):
            part = ''.join(map(lambda x: '$' if x in self.V_n else x, part))
            for rule in self.rules:
                right = ''.join(map(lambda x: '$' if x in self.V_n else x, rule.right))
                if part == right:
                    return rule.left
            return None
        while cur < length:
            step += 1
            priority = '<'
            cur_sym = sentence[cur]

            compare_sym = ''
```

```
            for i in range(len(stack)-1, -1, -1):
                if stack[i] in self.V_t:
                    priority = self.priority_table.get((stack[i], cur_sym))
                    compare_sym = stack[i]
                    break
            if priority == '>':
                action = 'reduce'
            elif priority == '<' or priority == '=':
                action = 'move in'
            else:
                raise OPGRunError(pos=cur+1)
            yield {'step': str(step),
                   'stack': ''.join(stack),
                   'priority': compare_sym+priority+cur_sym,
                   'current': cur_sym,
                   'left': sentence[cur+1:],
                   'action': action}
            if priority == '>':
                vt = None
                t = ''
                while True:
                    if stack[-1] in self.V_t:
                        if vt and self.priority_table.get((stack[-1], vt)) == '<':
                            break
                        else:
                            vt = stack[-1]
                    t += stack[-1]
                    stack.pop()
                    if len(stack) == 0:
                        break
                t = t[::-1]
                res = reduce(t)
                if res:
                    stack.append(res)
                else:
                    raise OPGRunError(pos=cur+1)
            else:
                stack.append(cur_sym)
                cur += 1

    def print_analyse(self, sentence):
        length = len(sentence)
        # printing template
        template = '{step:4}   {stack:sentence_length}   {priority:8}   {current:7}   ' \
                   '{left:sentence_length}   {action:8}'.replace('sentence_length', str(max(5,
length)))
        print(template.format(step='Step', stack='Stack', priority='Priority',
                              current='Current', left='Left', action='Action'))
        try:
            for item in self.analyse(sentence):
                print(template.format(step=item['step'],
                                      stack=item['stack'],

                                      priority=item['priority'],
```

```
                                    current=item['current'],
                                    left=item['left'],
                                    action=item['action']))
        except OPGRunError as e:
            print(e.message)

    def complete_analyse(self, sentence):
        res = list()
        try:
            res.append({
                'state': 'normal',
                'step': 'Step',
                'stack': 'Stack',
                'priority': 'Priority',
                'current': 'Current',
                'left': 'Left',
                'action': 'Action',
            })
            for item in self.analyse(sentence):
                item['state'] = 'normal'
                res.append(item)
        except OPGRunError as e:
            res.append({
                'state': 'error',
                'message': e.message
            })
        finally:
            return res


def main():
    opg = OPGEngine()
    with open('../doc/opg.txt') as f:
        grammar = f.read().split('\n')
    opg.get_rules(grammar)
    if opg.get_priority_table():
        opg.print_priority_table()
        sentence = '(i+i)'
        opg.print_analyse(sentence)


if __name__ == '__main__':
    main()
```

整个算符优先分析程序被封装成**OPGEnfine**类，**rules**为文法规则，**V_n**为非终结字符集，**V_t**为终结字符集，**priority_table**为优先关系矩阵。

get_rules根据输入的文法得到规则、非终结字符集和终结字符集，并任取一个不是非终结字符的大写字母**S**，假设文法入口为**E**，增加一条规则`S->#E#`，并相应在**V_n**里加入一个**S**，在**V_t**中加入一个**#**。

get_priority_table求得文法的优先关系矩阵，首先需要求得**FIRSTVT**和**LASTVT**，然后利用这两个集合遍历规则求得算符优先关系，若求得某两字符优先关系不唯一，则说明该文法不是算符优先文法，程序报错。

print_priority_table打印文法的优先关系矩阵。

complete_priority_table返回字典格式的优先关系矩阵，用于前端js处理显示。

analyse根据优先关系矩阵对输入句子进行分析，利用一个栈存储已分析字符，每次从栈顶开始找到第一个终结符**a**，判断该**a**与当前分析字符**b**的优先关系。若当**a**与**b**的优先关系无法判断时，说明出现了不合法的句子，程序报错，若`a<b`或`a=b`则将b进栈，若`a>b`则对栈内字符进行规约，从栈内字符集合找到最左素短语并进行规约，在这里进行规约的时候，非终结字符集对分析过程中寻找最左素短语没有影响，可以用任意占位符代替，规约时若无法找到规则，也说明出现不合法的句子，程序报错。

print_analyse打印分析过程。

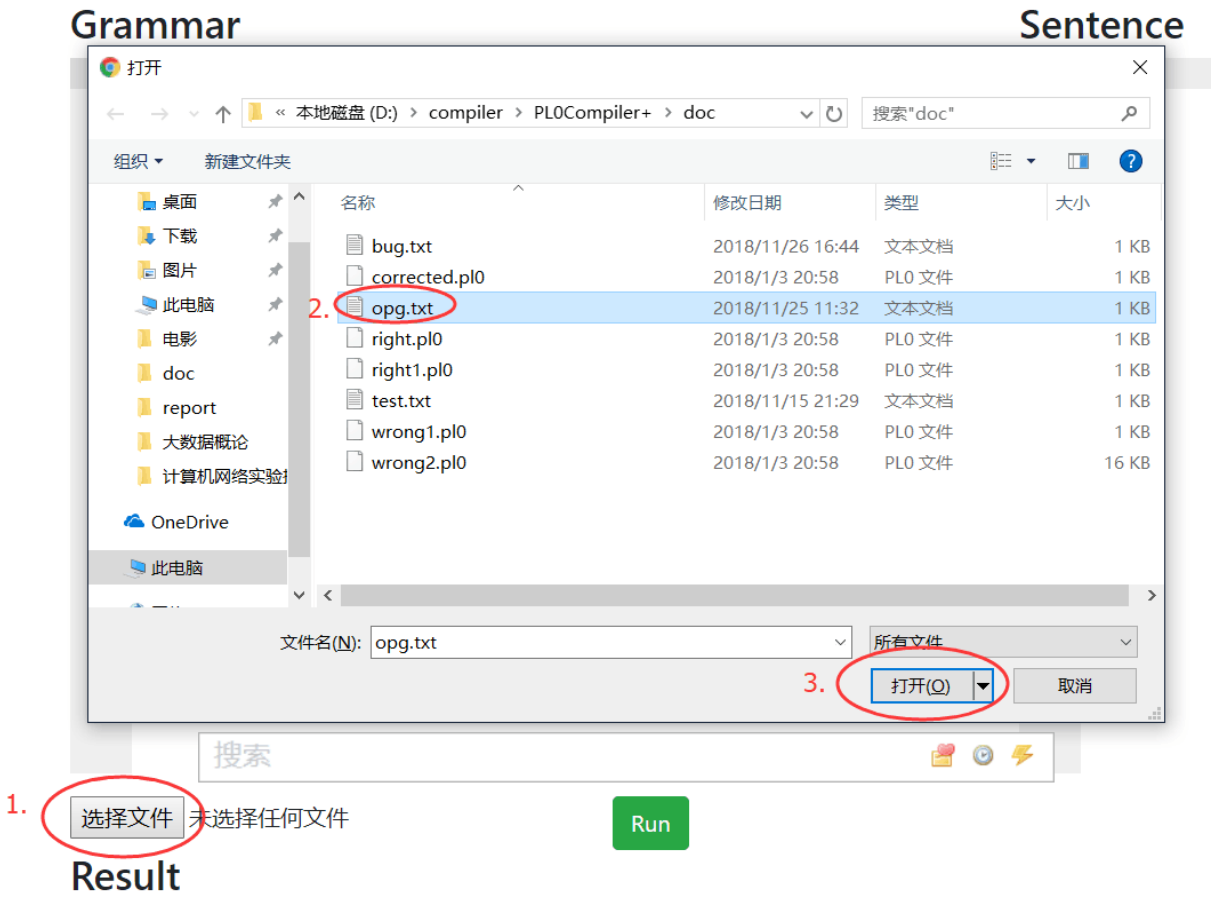
complete_analyse返回字典格式的分析过程，用于前端js处理显示。

程序说明

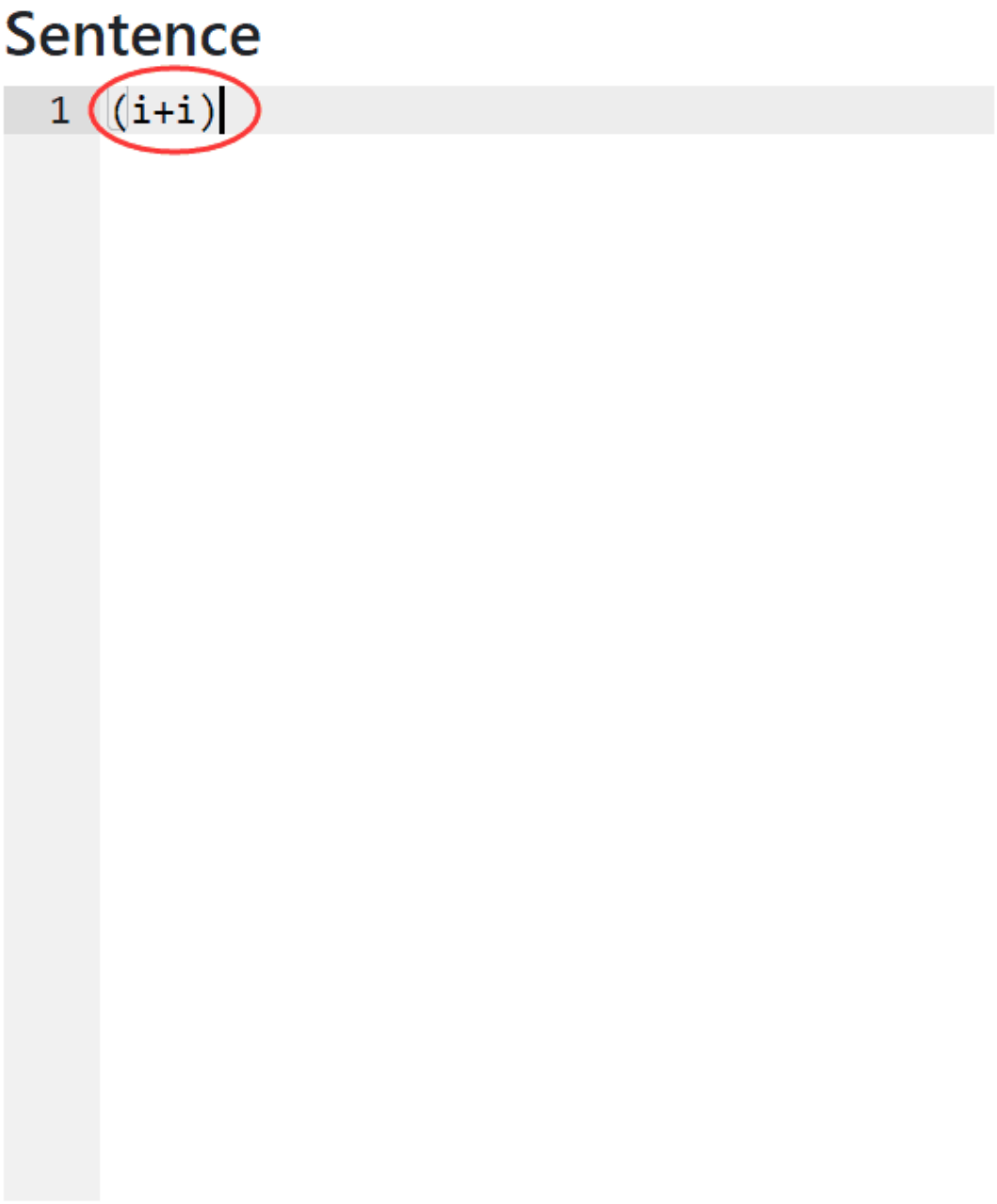
1. 文法的第一条规则必须为文法入口。
2. 每一条规则输入格式有严格规定，如`E->T*F|F`，终结符规定为大写字母，且终结符和非终结符都为单字符。不能出现扩展的BNF文法，即`{}、[]、()`带特殊含义的规则。 规则中不能有`#`，该字符用于辅助移进-归约判断。
3. 句子正常输入即可，无格式要求。

程序使用

1. 在Grammar输入框内输入文法，可手动输入也可文件导入。



2. 在Sentence输入框内输入句子。



3. 点击RUN按钮，下方Result会显示优先关系矩阵和移进规约的过程，若显示`Rus1t Error`说明出现错误， 可能是因为文法不是算符优先文法，也可能句子错误。

Result

Priority Table

	*	(	+	i	)	#
*	>	<	>	<	>	>
(	<	<	<	<	=	?
+	<	<	>	<	>	>
i	>	?	>	?	>	>
)	>	?	>	?	>	>
#	<	<	<	<	?	=

Procedure

Step	Stack	Priority	Current	Left	Action
1	#	#<(	(	i+i)#	move in
2	#(	(	i	+i)#	move in

4. 程序报错显示
   - 非算符优先文法

Result Error

Priority T It isn't OPG

It isn't OPG

   - 句子错误

Result Error

Priority Tab OPG runtime error in character 3

	*	(	+	i	)	#
*	>	<	>	<	>	>
(	<	<	<	<	=	?
+	<	<	>	<	>	>
i	>	?	>	?	>	>
)	>	?	>	?	>	>
#	<	<	<	<	?	=

Procedure

Step	Stack	Priority	Current	Left	Action
1	#	#	i	+b#	move in
2	#i	i>+	+	b#	reduce
3	#F	#<+	+	b#	move in

OPG runtime error in character 3

实验感想

通过本次实验，自己加深了对算符优先分析的理解，首先要从规则里获取非终结字符集合终结字符集，接着产生**FIRSTVT**和**LASTVT**集，根据这两个集合确定优先关系矩阵，之后根据这个矩阵重复寻找句子的最左素短语进行规约，在处理时将所有非终结符用同一占位符替换，实际上通过算符优先分析的句子不一定是符合文法的，想要判断是否符合文法还需要进行语义分析。在具体实现时，语法分析和语义处理结合起来进行，每进行一次规约就调用一次有关的语义处理程序，生成相应子表达式的代码，与归约为某个非终结符号相对应，语义程序分配一个工作单位来存放该子表达式的运算结果，因此对语义处理程序来说也不需要知道真正的非终结符的名字。