王浩算法实验报告

Honor Code

参考互联网实现中缀表达式读取

1.表达式解析

用户按照如下的符号输入表达式

```
原子命题 除了小写字母v之外的字母，大小写均可
析取 v
合取 ^
否定 !
蕴含 ->
双蕴含 <->
```

对输入的表达式进行预处理，去掉空格和非法符号

```
line.erase(remove_if(line.begin(),line.end(),[](char c) {
        return !(isalpha(c)||c=='^'||c=='!'||c=='-'||c=='<'||c=='>'||c=='('||c==')');
    }),line.end());
```

表达式存储的数据结构

```
struct tree {
    char val;
    tree *left;
    tree *right;
};
```

为了实现表达式读取，我们引入栈

```
stack<tree*> subTreeStack;
stack<char> operatorStack;
```

进行循环读入

- 对于原子命题，新建一棵树，压入`subTreeStack`

```
if(isalpha(line[i])&&line[i]!='v') {
    tree* node=new tree;
    node->val=line[i];
    node->left=nullptr;
    node->right=nullptr;
    subTreeStack.push(node);
}
```

- 对于连接词，不断弹出`operatorStack`中优先级更高的的连接词，组成一颗子树，弹出结束之后把自己压入operatorStack，其中注意否定连接词只有右子树，以`v`为例

```
if(line[i]=='v') {
    while(!operatorStack.empty()&&precedence(operatorStack.top())>=precedence(line[i])) {
        tree* node=new tree;
        node->val=operatorStack.top();
        node->right=subTreeStack.top();
        subTreeStack.pop();
        if(operatorStack.top()!='!') {
            node->left=subTreeStack.top();
        } else {
            node->left=nullptr;
        }
        subTreeStack.push(node);
        operatorStack.pop();
    }
    operatorStack.push(line[i]);
}
```

- 对于括号表达式，左括号直接压入`operatorStack`，右括号弹出之前的所有表达式直到左括号，最后弹出左括号

```
if(line[i]=='(') {
    operatorStack.push('(');
} else if(line[i]==')') {
    while(operatorStack.top()!='('&&!operatorStack.empty()) {
        tree* node=new tree;
        node->val=operatorStack.top();
        node->right=subTreeStack.top();
        subTreeStack.pop();
        if(operatorStack.top()!='!') {
            node->left=subTreeStack.top();
            subTreeStack.pop();
        } else {
            node->left=nullptr;
        }
        subTreeStack.push(node);
        operatorStack.pop();
    }
    if(!operatorStack.empty()) {
        operatorStack.pop();
    }
}
```

- 最后弹出`operatorStack`中剩下的连接词，组成子树，压入`subtreeStack`

```
while(!operatorStack.empty()) {
    tree* node=new tree;
    node->val=operatorStack.top();
    node->right=subTreeStack.top();
    subTreeStack.pop();
    if(operatorStack.top()!='!') {
        node->left=subTreeStack.top();
        subTreeStack.pop();
    } else {
        node->left=nullptr;
    }
    subTreeStack.push(node);
    operatorStack.pop();
}
```

其中定义操作优先级函数来实现连接词对比他优先级更高的连接词的弹出

```
int precedence(char c) {
    if(c=='!') return 4;
    if(c=='^') return 3;
    if(c=='v') return 2;
    if(c=='>') return 1;
    if(c=='<') return 0;
    return -1;
}
```

读取结束之后可以打印二叉树。为打印出树的形状，按照DFS遍历，先打印右子树，再打印节点，最后打印左子树，通过空格来保留树的形状

```
void printTree(tree* root,int depth=0) {
    if(root==nullptr) return;
    printTree(root->right,depth+1);
    for (int i=0;i<depth;++i) cout<<"   ";
    cout<<root->val<<endl;
    printTree(root->left,depth+1);
}
```

可以打印出含括号的中缀表达式，对遍历的每一层（除了第0层），先加入左括号，然后打印左子树，再打印节点，最后打印右子树，最后加上括号

```
void printExpression(tree* root,int depth=0) {
    if(root==nullptr) return;
    if((root->left!=nullptr||root->val=='!')&&depth!=0) {
        cout<<"(";
    }
    printExpression(root->left,depth+1);
    if(root->val=='>') {
        cout<<"->";
    } else if(root->val=='<') {
        cout<<"<->";
    } else {
        cout<<root->val;
    }
    printExpression(root->right,depth+1);
    if(root->right!=nullptr&&depth!=0) {
        cout<<")";
    }
}
```

其中应对错误输入的措施如下

- 如果需要访问`operatorStack`或者`subTreeStack`的地方，这两个栈为空，那么说明输入的表达式不正确，返回`nullptr`。
- 如果输入的`<->`和`->`没有按照规范写，返回`nullptr`
- 如果最后`subTreeStack`有超过一个元素，说明输入的表达式不正确，返回`nullptr`
- 最后，如果得到的表达式为`nullptr`，那么说明输入的表达式为空，`cout<<"Empty Input or Not Well-Formed Formula";`

2.王浩算法实现

主函数的逻辑

```
bool whalgorithm(vector<tree*>& head,vector<tree*>& tail)
```

对前缀表达式`vector<tree*> head` 和 `vector<tree*> tail` 进行处理

先处理后缀表达式

以后键中的`v`为例，按照王浩算法的变形规则，把`v`命题的左右子命题都加入到后键的`vector`中，然后擦除`v`命题

```
if(c=='v') {
    vector<tree*> temp1=head;
    vector<tree*> temp2=tail;
    temp2.push_back(tail[i]->left);
    temp2.push_back(tail[i]->right);
    temp2.erase(temp2.begin()+i);
}
```

再用完全相同的方法处理前缀表达式

其中，每一步处理之后都递归处理新的表达式，对于不能够证明的命题，直接结束推理

```
if(!whalgorithm(temp1,temp2)) {
    return false;
}
```

处理新表达式结束之后，检查是否到达边界（也就是是否已经完全化简）

```
else if(backFrontCheck(head)&&backBackCheck(tail,i)) {
    return true;
}
```

其中处理边界的函数如下。前键的边界检查函数表示，当前键中所有的元素都是非`v`的字母（也就是原子命题），那么表达式化简已经完成；后键的边界检查函数表示，当后键从这一位开始，后面所有的元素都是非`v`的字母（也就是原子命题），那么表达式化简已经完成。由于之前已经递归证明过这是一个定理，这里就可以返回`true`了

```
bool backFrontCheck(vector<tree*> head) {
    for(int i=0;i<head.size();++i) {
        if(!isalpha(head[i]->val)||head[i]->val=='v') {
            return false;
        }
    }
    return true;
}
bool backBackCheck(vector<tree*> tail,int i) {
    for(int j=i+1;j<tail.size();++j) {
        if(!isalpha(tail[j]->val)||tail[j]->val=='v') {
            return false;
        }
    }
    return true;
}
```

当全部化简为原子命题之后，计算是否有相同字符。首先将`vector`中的内容拷贝到`string`中，将`string`按字典序排序，将公共部分放入`set`中，如果`set`为空，表明这个命题不是axiom，无法证明，返回`false`

```
string compare1;
string compare2;
for(int j=0;j<head.size();++j) {
    compare1+=head[j]->val;
}
for(int j=0;j<tail.size();++j) {
    compare2+=tail[j]->val;
}
sort(compare1.begin(),compare1.end());
sort(compare2.begin(),compare2.end());
vector<char> intersection;
set_intersection(compare1.begin(),compare1.end(),compare2.begin(),compare2.end(),back_ins
erter(intersection));
if(!intersection.empty()) {
    cout<<" --- ";
    cout<<compare1<<" | "<<compare2<<" ---";
    cout<<"TRUE"<<endl;
    return true;
} else {
    cout<<" --- ";
    cout<<compare1<<" | "<<compare2<<" ---";
    cout<<"FALSE"<<endl;
    return false;
}
```

3.实验检测

分三种情况进行测试

- 错误输入
    - 括号输入错误

      ```
      ava))
      Processed input: ava))
      Empty Input or Not Well-Formed Formula
      ```

    - 连续输入字母

      ```
      av(bbvb)
      Processed input: av(bbvb)
      Empty Input or Not Well-Formed Formula
      ```

    - 连续输入2元连接词

      ```
      a-->a
      Processed input: a-->a
      Empty Input or Not Well-Formed Formula
      ```

    - 蕴含输入形式错误

      ```
      S->c^^b
      Processed input: S->c^^b
      Empty Input or Not Well-Formed Formula
      ```

    - 处理时自动删除非法字符，然后进行处理

      ```
      a$va^c
      Processed input: ava^c
      Binary Tree Constructed:
          c
        ^
          a
      v
        a
      Expression Constructed:
      av(a^c)
      ```

- 重言式（正确的命题）

为了不让这个文档太长，非常长的证明放在了另外的`log`文档中。都是非常好命题，使我的Typora无法动弹。

下面把一些非常简单的书上的命题放在文档中。

```
(P->(Q->R))<->((P^Q)->R)
Processed input: (P->(Q->R))<->((P^Q)->R)
Binary Tree Constructed:
```

```
    R
  >
      Q
    ^
      P
<
      R
    >
      Q
  >
    P
Expression Constructed:
(P->(Q->R))<->((P^Q)->R)
Start Reasoning:
***  => (P->(Q->R))<->((P^Q)->R)
*** P->(Q->R) => (P^Q)->R
*** P->(Q->R),P^Q => R
*** P^Q,Q->R => R
*** Q->R,P,Q => R
*** P,Q,R => R
 --- PQR | R ---TRUE
*** P^Q,R => R
*** R,P,Q => R
 --- PQR | R ---TRUE
*** P^Q => R,P
*** P,Q => R,P
 --- PQ | PR ---TRUE
*** P->(Q->R),P,Q => R
*** P,Q,Q->R => R
*** P,Q,R => R
 --- PQR | R ---TRUE
*** Q->R => (P^Q)->R
*** Q->R,P^Q => R
*** P^Q,R => R
*** R,P,Q => R
 --- PQR | R ---TRUE
*** P^Q => R,Q
*** P,Q => R,Q
 --- PQ | QR ---TRUE
*** Q->R,P,Q => R
*** P,Q,R => R
 --- PQR | R ---TRUE
*** R => (P^Q)->R
*** R,P^Q => R
*** R,P,Q => R
 --- PQR | R ---TRUE
*** (P^Q)->R => P->(Q->R)
*** (P^Q)->R,P => Q->R
*** (P^Q)->R,P,Q => R
*** P,Q,R => R
 --- PQR | R ---TRUE
*** P,R => Q->R
*** P,R,Q => R
 --- PQR | R ---TRUE
*** R => P->(Q->R)
*** R,P => Q->R
*** R,P,Q => R
 --- PQR | R ---TRUE
It's true!
```

```
((P->Q)^(P->!Q))<->!P
Processed input: ((P->Q)^(P->!Q))<->!P
Binary Tree Constructed:
    P
  !
<
        Q
      !
    >
      P
  ^
      Q
    >
      P
Expression Constructed:
((P->Q)^(P->(!Q)))<->(!P)
Start Reasoning:
***  => ((P->Q)^(P->(!Q)))<->(!P)
*** (P->Q)^(P->(!Q)) => !P
*** (P->Q)^(P->(!Q)),P =>
*** P,P->Q,P->(!Q) =>
*** P,P->(!Q),Q =>
*** P,Q,!Q =>
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P,P->(!Q) => P
*** P,!Q => P
*** P => P,Q
 --- P | PQ ---TRUE
*** P,P->Q,!Q =>
*** P,!Q,Q =>
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P,!Q => P
*** P => P,Q
 --- P | PQ ---TRUE
*** P,P->Q => Q
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P->Q,P->(!Q) => !P
*** P->Q,P->(!Q),P =>
*** P->(!Q),P,Q =>
*** P,Q,!Q =>
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P->(!Q),P => P
*** P,!Q => P
*** P => P,Q
 --- P | PQ ---TRUE
*** P->Q,P,!Q =>
*** P,!Q,Q =>
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P,!Q => P
*** P => P,Q
 --- P | PQ ---TRUE
*** P->Q,P => Q
*** P,Q => Q
 --- PQ | Q ---TRUE
*** P->(!Q),Q => !P
*** P->(!Q),Q,P =>
*** Q,P,!Q =>
*** Q,P => Q
 --- PQ | Q ---TRUE
*** Q,!Q => !P
*** Q,!Q,P =>
*** Q,P => Q
 --- PQ | Q ---TRUE
*** Q => !P,Q
*** Q,P => Q
 --- PQ | Q ---TRUE
*** P->(!Q) => !P,P
*** P->(!Q),P => P
*** P,!Q => P
*** P => P,Q
 --- P | PQ ---TRUE
*** !Q => !P,P
*** !Q,P => P
*** P => P,Q
 --- P | PQ ---TRUE
***  => !P,P,Q
*** P => P,Q
 --- P | PQ ---TRUE
*** P->Q,!Q => !P
*** P->Q,!Q,P =>
*** !Q,P,Q =>
*** P,Q => Q
 --- PQ | Q ---TRUE
*** !Q,P => P
*** P => P,Q
 --- P | PQ ---TRUE
*** P->Q,P => Q
*** P,Q => Q
 --- PQ | Q ---TRUE
*** !Q,Q => !P
*** !Q,Q,P =>
*** Q,P => Q
 --- PQ | Q ---TRUE
*** Q => !P,Q
*** Q,P => Q
 --- PQ | Q ---TRUE
*** !Q => !P,P
*** !Q,P => P
*** P => P,Q
 --- P | PQ ---TRUE
***  => !P,P,Q
*** P => P,Q
 --- P | PQ ---TRUE
*** P->Q => !P,Q
*** P->Q,P => Q
*** P,Q => Q
 --- PQ | Q ---TRUE
*** Q => !P,Q
*** Q,P => Q
 --- PQ | Q ---TRUE
*** !P => (P->Q)^(P->(!Q))
*** !P => P->Q
*** !P,P => Q
*** P => Q,P
 --- P | PQ ---TRUE
***  => P->Q,P
*** P => P,Q
 --- P | PQ ---TRUE
*** !P => P->(!Q)
*** !P,P => !Q
*** !P,P,Q =>
*** P,Q => P
 --- PQ | P ---TRUE
*** P => !Q,P
*** P,Q => P
 --- PQ | P ---TRUE
***  => P->(!Q),P
*** P => P,!Q
*** P,Q => P
 --- PQ | P ---TRUE
***  => (P->Q)^(P->(!Q)),P
***  => P,P->Q
*** P => P,Q
 --- P | PQ ---TRUE
***  => P,P->(!Q)
*** P => P,!Q
*** P,Q => P
 --- PQ | P ---TRUE
It's true!
```

```
(P<->Q)^(Q<->R)->(P<->R)
```

```
Processed input: (P<->Q)^(Q<->R)->(P<->R)
Binary Tree Constructed:
    R
  <
    P
>
      R
    <
      Q
  ^
      Q
    <
      P
Expression Constructed:
((P<->Q)^(Q<->R))->(P<->R)
Start Reasoning:
***  => ((P<->Q)^(Q<->R))->(P<->R)
*** (P<->Q)^(Q<->R) => P<->R
*** (P<->Q)^(Q<->R),P => R
*** P,P<->Q,Q<->R => R
*** P,Q<->R,P,Q => R
*** P,P,Q,Q,R => R
 --- PPQQR | R ---TRUE
*** P,Q<->R => R,P,Q
*** P,Q,R => R,P,Q
 --- PQR | PQR ---TRUE
*** P,P<->Q,Q,R => R
*** P,Q,R,P,Q => R
 --- PPQQR | R ---TRUE
*** (P<->Q)^(Q<->R),R => P
*** R,P<->Q,Q<->R => P
*** R,Q<->R,P,Q => P
*** R,P,Q,Q,R => P
 --- PQQRR | P ---TRUE
*** R,Q<->R => P,P,Q
*** R,Q,R => P,P,Q
 --- QRR | PPQ ---TRUE
*** R,P<->Q,Q,R => P
*** R,Q,R,P,Q => P
 --- PQQRR | P ---TRUE
*** P<->Q,Q<->R => P<->R
*** P<->Q,Q<->R,P => R
*** Q<->R,P,P,Q => R
*** P,P,Q,Q,R => R
 --- PPQQR | R ---TRUE
*** Q<->R,P => R,P,Q
*** P,Q,R => R,P,Q
 --- PQR | PQR ---TRUE
*** P<->Q,P,Q,R => R
*** P,Q,R,P,Q => R
 --- PPQQR | R ---TRUE
*** P<->Q,Q<->R,R => P
*** Q<->R,R,P,Q => P
*** R,P,Q,Q,R => P
 --- PQQRR | P ---TRUE
*** Q<->R,R => P,P,Q
*** R,Q,R => P,P,Q
 --- QRR | PPQ ---TRUE
*** P<->Q,R,Q,R => P
*** R,Q,R,P,Q => P
 --- PQQRR | P ---TRUE
*** Q<->R,P,Q => P<->R
*** Q<->R,P,Q,P => R
*** P,Q,P,Q,R => R
 --- PPQQR | R ---TRUE
*** Q<->R,P,Q,R => P
*** P,Q,R,Q,R => P
 --- PQQRR | P ---TRUE
*** P,Q,Q,R => P<->R
*** P,Q,Q,R,P => R
 --- PPQQR | R ---TRUE
*** P,Q,Q,R,R => P
 --- PQQRR | P ---TRUE
*** Q<->R => P<->R,P,Q
*** Q<->R,P => P,Q,R
*** P,Q,R => P,Q,R
 --- PQR | PQR ---TRUE
*** Q<->R,R => P,Q,P
*** R,Q,R => P,Q,P
 --- QRR | PPQ ---TRUE
*** Q,R => P<->R,P,Q
*** Q,R,P => P,Q,R
 --- PQR | PQR ---TRUE
*** Q,R,R => P,Q,P
 --- QRR | PPQ ---TRUE
*** P<->Q,Q,R => P<->R
*** P<->Q,Q,R,P => R
*** Q,R,P,P,Q => R
 --- PPQQR | R ---TRUE
*** P<->Q,Q,R,R => P
*** Q,R,R,P,Q => P
 --- PQQRR | P ---TRUE
*** Q,R,P,Q => P<->R
*** Q,R,P,Q,P => R
 --- PPQQR | R ---TRUE
*** Q,R,P,Q,R => P
 --- PQQRR | P ---TRUE
It's true!
```

- 非重言式（无法证明的命题）

同样的过长的过程放在了另外的文档中

```
a->a^b
```

```
Processed input: a->a^b
Binary Tree Constructed:
    b
  ^
    a
>
  a
Expression Constructed:
a->(a^b)
Start Reasoning:
***  => a->(a^b)
*** a => a^b
*** a => a
 --- a | a ---TRUE
*** a => b
 --- a | b ---FALSE
It's false.
```

```
(P<->Q)^(Q<->R)<->(P<->R)
Processed input: (P<->Q)^(Q<->R)<->(P<->R)
Binary Tree Constructed:
    R
  <
    P
<
      R
    <
      Q
  ^
      Q
    <
      P
Expression Constructed:
((P<->Q)^(Q<->R))<->(P<->R)
Start Reasoning:
***  => ((P<->Q)^(Q<->R))<->(P<->R)
*** (P<->Q)^(Q<->R) => P<->R
*** (P<->Q)^(Q<->R),P => R
*** P,P<->Q,Q<->R => R
*** P,Q<->R,P,Q => R
*** P,P,Q,Q,R => R
 --- PPQQR | R ---TRUE
*** P,Q<->R => R,P,Q
*** P,Q,R => R,P,Q
 --- PQR | PQR ---TRUE
*** P,P<->Q,Q,R => R
*** P,Q,R,P,Q => R
 --- PPQQR | R ---TRUE
*** (P<->Q)^(Q<->R),R => P
*** R,P<->Q,Q<->R => P
*** R,Q<->R,P,Q => P
*** R,P,Q,Q,R => P
 --- PQQRR | P ---TRUE
*** R,Q<->R => P,P,Q
*** R,Q,R => P,P,Q
 --- QRR | PPQ ---TRUE
*** R,P<->Q,Q,R => P
*** R,Q,R,P,Q => P
 --- PQQRR | P ---TRUE
*** P<->Q,Q<->R => P<->R
*** P<->Q,Q<->R,P => R
*** Q<->R,P,P,Q => R
*** P,P,Q,Q,R => R
 --- PPQQR | R ---TRUE
*** Q<->R,P => R,P,Q
*** P,Q,R => R,P,Q
 --- PQR | PQR ---TRUE
*** P<->Q,P,Q,R => R
*** P,Q,R,P,Q => R
 --- PPQQR | R ---TRUE
*** P<->Q,Q<->R,R => P
*** Q<->R,R,P,Q => P
*** R,P,Q,Q,R => P
 --- PQQRR | P ---TRUE
*** Q<->R,R => P,P,Q
*** R,Q,R => P,P,Q
 --- QRR | PPQ ---TRUE
*** P<->Q,R,Q,R => P
*** R,Q,R,P,Q => P
 --- PQQRR | P ---TRUE
*** Q<->R,P,Q => P<->R
*** Q<->R,P,Q,P => R
*** P,Q,P,Q,R => R
 --- PPQQR | R ---TRUE
*** Q<->R,P,Q,R => P
*** P,Q,R,Q,R => P
 --- PQQRR | P ---TRUE
*** P,Q,Q,R => P<->R
*** P,Q,Q,R,P => R
 --- PPQQR | R ---TRUE
*** P,Q,Q,R,R => P
 --- PQQRR | P ---TRUE
*** Q<->R => P<->R,P,Q
*** Q<->R,P => P,Q,R
*** P,Q,R => P,Q,R
 --- PQR | PQR ---TRUE
*** Q<->R,R => P,Q,P
*** R,Q,R => P,Q,P
 --- QRR | PPQ ---TRUE
*** Q,R => P<->R,P,Q
*** Q,R,P => P,Q,R
 --- PQR | PQR ---TRUE
*** Q,R,R => P,Q,P
```

```
 --- QRR | PPQ ---TRUE
*** P<->Q,Q,R => P<->R
*** P<->Q,Q,R,P => R
*** Q,R,P,P,Q => R
 --- PPQQR | R ---TRUE
*** P<->Q,Q,R,R => P
*** Q,R,R,P,Q => P
 --- PQQRR | P ---TRUE
*** Q,R,P,Q => P<->R
*** Q,R,P,Q,P => R
 --- PPQQR | R ---TRUE
*** Q,R,P,Q,R => P
 --- PQQRR | P ---TRUE
*** P<->R => (P<->Q)^(Q<->R)
*** P<->R => P<->Q
*** P<->R,P => Q
*** P,P,R => Q
 --- PPR | Q ---FALSE
It's false.
```

```
(P->(Q->R))<->((PvQ)->R)
Processed input: (P->(Q->R))<->((PvQ)->R)
Binary Tree Constructed:
    R
  >
      Q
    v
      P
<
      R
    >
      Q
  >
    P
Expression Constructed:
(P->(Q->R))<->((PvQ)->R)
Start Reasoning:
***  => (P->(Q->R))<->((PvQ)->R)
*** P->(Q->R) => (PvQ)->R
*** P->(Q->R),PvQ => R
*** PvQ,Q->R => R
*** Q->R,P => R
*** P,R => R
 --- PR | R ---TRUE
*** Q->R,Q => R
*** Q,R => R
 --- QR | R ---TRUE
*** PvQ,R => R
*** R,P => R
 --- PR | R ---TRUE
*** R,Q => R
```

```
 --- QR | R ---TRUE
*** PvQ => R,P
*** P => R,P
 --- P | PR ---TRUE
*** Q => R,P
 --- Q | PR ---FALSE
It's false.
```