

离散数学实验报告

班 级： 软件工程191

学号姓名： 1190204104 周一魏

学 期： 2020-2021-1

### 实验一 构造合式公式的真值表

#### 一、实验目的

熟悉逻辑运算否定、合取、析取、蕴含、等价规则，利用程序语言实现命题公式的真值表运算，使其与命题的等值演算结果相同。

使学生熟练掌握利用计算机语言实现逻辑运算的基本方法。

#### 二、实验内容

对给出的任意一个合式公式（不超过四个命题变元），使学生会用C语言的程序编程表示出来，并且能够计算它在各组真值指派下所应有的真值，画出其真值表。

#### 三、 实验过程

## 1. 基本原理

（1）逻辑联结词的定义方法

逻辑连接词“非”：设p为命题, 复合命题 “非p”(或 “p的否定”)称为p的否定式, 记作¬p。符号¬称作否定联结词,其真值是这样定义的：若 P 的真值是T,那么┐P真值是F；若P的真值是F,则┐P 的真值是T。

逻辑连接词“合取”：设p ,q为二命题, 复合命题“p并且q”(或“p与q”)称为p与q的合取式, 记作p∧q, ∧称作合取联结词。它的真值是这样定义的：当且仅当 P 和 q 的真值都为 T 时，p∧q 的真值才为 T，否则p∧q 的真值为 F。

逻辑连接词“析取”：设 p ,q为命题, 复合命题“p或q”称作p与q的析取式,记作p∨q, ∨称作析取联结词。它的真值是这样定义的：当且仅当p和q的真值都为假（F）时，p∨q 的真值才为 F，否则p∨q 的真值为 T。

逻辑连接词“蕴涵”：设 p ,q为二命题, 复合命题 “如果p,则q” 称作p与q的蕴涵式, 记作p→q. →称作蕴涵联结词，运算对象p叫做前提（premise）、假设(Hypothesis)或前件，而q叫做结论(Conclusion)或后件。它的真值是这样定义的：当且仅当 p 是T ，而 q 是 F 时，p→ q 的真值才为 F，否则p→q的真值为 T。

逻辑连接词“等价”：设p, q为命题, 复合命题 “p当且仅当q”称作p与q的等价式, 记作p↔q, ↔称作等价联结词。它的真值是这样定义的：当且仅当 p 和 q 有相同的真值时，p ↔ q 的真值才为 T，否则 p ↔ q 的真值为 F。

（2）合式公式的表示方法

复合命题：由简单命题通过联结词联结而成的新命题叫复合命题。

合式公式(命题公式,公式)递归定义如下：

1. 单个命题常项或变项是合式公式,并称作原子合式公式；
2. 若A是合式公式, 则 (¬A)也是合式公式；
3. 若A, B是合式公式, 则(A∧B), (A∨B), (A→B), (A↔B)也是合式公式；
4. 只有有限次地应用(1)~(3)形成的符号串才是合式公式。

给出任意一个合式公式，我们可以将它用C程序表示出来，并且能够计算它在各组真值指派下所应有的真值（或是逻辑运算的结果）。这有多种方法。上面我们已经给出了逻辑连结词的定义，根据这种定义方法，我们也可以把一个合式公式表示成为条件语句中的条件表达式，这样我们就可以得到该合式公式的逻辑运算结果了。

（3）任意合式公式的真值表

真值：一个命题的真或假称为命题的真值。

真值表:命题公式在所有可能的赋值下的取值的列表，含n个变项的公式有2n个赋值。

构造真值表有如下约定：

1. 命题变元按字典序排列；
2. 对公式的每个解释，以二进制数从小到大或者从大到小顺序列出；
3. 若公式复杂，可先列出各子公式的真值(若有括号,则应从里层向外层展开)，最后列出所给公式的真值。

## 2. 程序代码

#include <bits/stdc++.h>

using namespace std;

int tag[30];

vector<bool> g;

void menu(){

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 欢迎进入逻辑运算软件 \*\*\n";

cout << "\*\* (可运算真值表,支持括号) \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 用!表示取反 \*\*\n";

cout << "\*\* 用&表示合取 \*\*\n";

cout << "\*\* 用|表示析取 \*\*\n";

cout << "\*\* 用^表示蕴含 \*\*\n";

cout << "\*\* 用~表示等值 \*\*\n";

cout << "\*\* 优先级:取反>合取>析取>蕴含>等价 \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

}

//中缀转后缀

string change(string s){

string ss="";

stack<char> st;

char t;

for(auto x:s){

if(x=='!'){

st.push(x);

}

else if(x=='&'){

if(st.empty()){

st.push(x);

}

else{

do{

t=st.top();

st.pop();

if(t=='('||t=='|'||t=='^'||t=='~'){

st.push(t);

}

else{

ss+=t;

}

}while(!st.empty()&&t!='('&&t!='|'&&t!='^'&&t!='~');

st.push(x);

}

}

else if(x=='|'){

if(st.empty()){。

st.push(x);

}

else{

do{

t=st.top();

st.pop();

if(t=='('||t=='^'||t=='~'){

st.push(t);

}

else{

ss+=t;

}

}while(!st.empty()&&t!='('&&t!='^'&&t!='~');

st.push(x);

}

}

else if(x=='^'){

if(st.empty()){

st.push(x);

}

else{

do{

t=st.top();

st.pop();

if(t=='('||t=='~'){

st.push(t);

}

else{

ss+=t;

}

}while(!st.empty()&&t!='('&&t!='~');

st.push(x);

}

}

else if(x=='~'){

if(st.empty()){

st.push(x);

}

else{

do{

t=st.top();

st.pop();

if(t=='('){

st.push(t);

}

else{

ss+=t;

}

}while(!st.empty()&&t!='(');

st.push(x);

}

}

else if(x=='('){

st.push(x);

}

else if(x==')'){

t=st.top();

st.pop();

while(t!='('){

ss+=t;

t=st.top();

st.pop();

}

}

else if(x>='a'&&x<='z'){

ss+=x;

}

else{

cout<<"非法\n";

exit(0);

}

}

while(!st.empty()){

t=st.top();

st.pop();

ss+=t;

}

char temp=ss[ss.length()-1];

if(temp>='a'&&temp<='z') return ss;

else {

cout<<"非法\n";

exit(0);

}

}

bool calc(string s){

bool ans;

stack<bool> st;

for(auto c:s){

if(c=='!'){

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t=st.top();

st.pop();

st.push(!t);

}

else if(c=='&'){

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t1=st.top();

st.pop();

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t2=st.top();

st.pop();

st.push(t1&t2);

}

else if(c=='|'){

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t1=st.top();

st.pop();

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t2=st.top();

st.pop();

st.push(t1|t2);

}

else if(c=='^'){

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t1=st.top();

st.pop();

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t2=st.top();

st.pop();

st.push((!t1)|t2);

}

else if(c=='~'){

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t1=st.top();

st.pop();

if(st.empty()){

cout<<"非法\n";

exit(0);

}

bool t2=st.top();

st.pop();

st.push(t1==t2);

}

else{

st.push(g[tag[c-'a']]);

}

}

return st.top();

}

int main(){

menu();

string s,ss;

cin>>s;

ss=change(s);

cout<<ss<<"\n";

vector<char> v;

int cnt=0;

for(auto c:s){

if(isalpha(c)){

if(!tag[c-'a']){

tag[c-'a']=cnt++;

v.push\_back(c);

}

}

}

for(auto x:v) cout<<x<<"\t";

cout<<s<<"\n";

for(int i=0;i<(1<<cnt);i++){

int t=i;

g.clear();

for(int j=0;j<cnt;j++){

g.push\_back(t%2);

t/=2;

}

for(int x:g) cout<<x<<"\t";

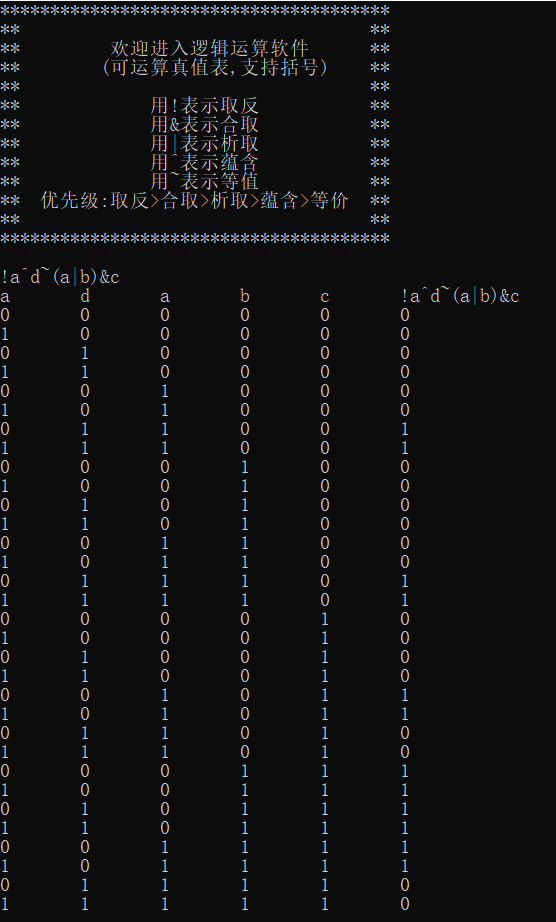
cout<<calc(ss)<<"\n";

}

return 0;

}

# 3.实验结果及分析



#### 四、实验总结

对给出的任意一个合式公式（不超过四个命题变元），我们用计算机语言的语句表示出来，并且计算出了它在各组真值指派下所应有的真值。通过这个实验使我们加深了对合适公示以及逻辑运算的理解和中缀表达式和后缀表达式的使用和转换。

### 实验二 关系的闭包运算

#### 一、实验目的

通过算法设计并编程实现求给定关系的各种闭包运算，加深学生对闭包运算的概念的理解。

#### 二、实验内容

给定关系R，求R的自反闭包、对称闭包及R的传递闭包。

#### 三、 实验过程

若关系R的关系矩阵为M，而自反闭包为A（即r（R）=A），对称闭包为B（即S（R）=B），则A=M∨I B=M∨MT 其中，I为恒等矩阵，MT为M的转置矩阵。

## 2. 程序代码

#include <bits/stdc++.h>

using namespace std;

const int N=105;

int n;

int a[N][N],b[N][N];

void Warshall(){

for(int j=1;j<=n;j++){ //第1列到最后列

for(int i=1;i<=n;i++){ //第j列从第1行到最后行

if(b[i][j]){

for(int k=1;k<=n;k++) {

b[i][k]|=b[j][k];

}

}

}

}

}

int main(){

puts("请输入矩阵大小n");

cin>>n;

puts("请输入n\*n的矩阵");

bool flag=1;

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

cin>>a[i][j];

if(a[i][j]>1&&a[i][j]<0) flag=0;

}

}

if(!flag){

printf("输入格式有误\n");

}

else{

puts("自反闭包：");

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

b[i][j]=a[i][j];

}

}

for(int i=1;i<=n;i++){

b[i][i]=1;

}

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

cout<<b[i][j]<<" ";

}

puts("");

}

puts("-------------------");

puts("对称闭包：");

memset(b,0,sizeof b);

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

if(a[i][j]) b[i][j]=b[j][i]=1;

}

}

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

cout<<b[i][j]<<" ";

}

puts("");

}

puts("-------------------");

puts("传递闭包：");

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

b[i][j]=a[i][j];

}

}

Warshall();

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

cout<<b[i][j]<<" ";

}

puts("");

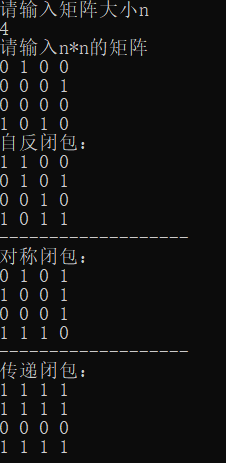
}

}

return 0;

}

# 3.实验结果及分析



#### 四、实验总结

学会了Warshall算法，熟悉自反闭包、对称闭包及传递闭包。

### 实验三 最小生成树的Kruskal算法

#### 一、实验目的

通过算法设计并编程实现求出给定无向连通加权图的一棵最小生成树，加深学生对求最小生成树的Kruskal算法的理解。

#### 二、实验内容

给定无向连通加权图，编程设计求出其一棵最小生成树。

#### 三、 实验过程

设所给定无向连通加权图具有n个结点，m条边，首先，将各条边的权按从小到大的顺序排序。然后依次将这些边按所给图的结构放到生成树中去。如果在放置某一条边时，使得生成树形成回路，则删除这条边。这样，直至生成树具有n-1条边时，我们所得到的就是一棵最小生成树。

## 2. 程序代码

#include <bits/stdc++.h>

using namespace std;

const int N=2e5+10,INF=0x3f3f3f3f;

int n,m,fa[N],depth[N];

struct edge{

int u,v,w;

}e[N];

bool cmp(edge x,edge y){

return x.w<y.w;

}

void init(int n){

for(int i=1;i<=n;i++)

fa[i]=i,depth[i]=1;

}

//查询树的根

int find(int x){

if(x!=fa[x]) fa[x]=find(fa[x]);

return fa[x];

}

//合并a和b所属的集合

void unite(int a,int b){

a=find(a),b=find(b);

if(depth[a]==depth[b]){

depth[a]=depth[a]+1;

fa[b]=a;

}

else{

if(depth[a]<depth[b]) fa[a]=b;

else fa[b]=a;

}

}

//判断a和b是否属于同一个集合

bool same(int a,int b){

return find(a)==find(b);

}

int Kruskal(){

int res=0,count=0;

for(int i=0;i<m;i++){

if(!same(e[i].u,e[i].v)){

unite(e[i].u,e[i].v);

res+=e[i].w;

count++;

}

if(count==n-1) return res;

}

return -1;

}

int main(){

puts("请输入结点数n和边数m");

cin>>n>>m;

puts("请依次输入m条无向边连接结点u,v和长度w");

for(int i=0;i<m;i++){

scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);

}

sort(e,e+m,cmp);

init(n);

int k=Kruskal();

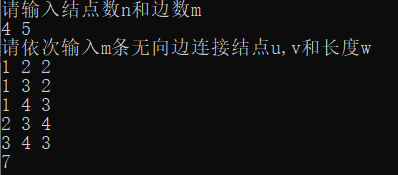
if(k!=-1) cout<<k;

else puts("impossible");

return 0;

}

# 3.实验结果及分析



#### 四、实验总结

加深对求最小生成树的Kruskal算法的理解以及并查集和贪心算法的应用。