به نام خدا

پروژه ساده‌سازی توابع با روش QM

درس مدار منطقی

استاد فرهادی

یگانه رضوانی

۴۰۳۱۸۶۶۳

اسما فاریابی

۴۰۳۲۴۱۰۳

رامینا قنبر آبادی

۴۰۱۲۷۳۷۳

فهرست مطالب:

معرفی پروژه:

این پروژه یک پیاده‌سازی کامل از الگوریتم کویین-مک‌کلاسکی برای ساده‌سازی توابع بولی ارائه می‌دهد. این برنامه از ورودی به صورت عبارت بولی پشتیبانی می‌کند و برای استفاده در محیط‌های آموزشی و تحقیقاتی طراحی شده است.

ویژگی‌های کلیدی:

- اعتبارسنجی عبارات ورودی
- قابلیت پردازش عبارات با پرانتزهای تو در تو
- محاسبه نتیجه برای تمام ترکیبات ممکن
- پشتیبانی از عملگرهای استاندارد بولی

متدهای اصلی:

- checkValidation
- findMin
- foundPI
- foundEPI

❖ متد checkValidation:

این متد، اعتبارسنجی کامل عبارات بولی ورودی را چک می‌کند و تمام قواعد نحوی عبارت بولی را بررسی می‌کند.

پارامترهای ورودی این متد:

- input از نوع string که همان عبارت بولی ورودی به عنوان آرگومان به آن داده می‌شود.
- number از نوع integer که تعداد متغیرهای مجاز برای بررسی محدوده A-Z است.

اگر عبارت معتبر باشد، خروجی متد true و اگر نامعتبر باشد false است.

موارد بررسی شده:

- توازن پرانتزها
- ترتیب صحیح عملگرها و عملوندها
- عدم وجود کاراکترهای نامعتبر
- رعایت محدوده متغیرها

❖ متد findMin:

این متد برای ارزیابی عبارت بولی با استفاده از روش پشته ای است که مقدار عبارت را برای یک مقدار خاص (یک مینترم) محاسبه می‌کند.

پارامترهای ورودی این متد:

- expression از نوع string که همان عبارت بولی ورودی به عنوان آرگومان به آن داده می‌شود.
- binary از نوع string که رشته بیتی است که مقادیر متغیرها را مشخص می‌کند.

این متد نتیجه ارزیابی را به صورت true یا false بر می‌گرداند.

نحوه کار:

- از دو پشته برای عملوندها و عملگرها استفاده می‌کند.
- عبارت را به صورت کاراکتر به کاراکتر پردازش می‌کند.
- از الگوریتم shunting-yard برای محاسبه استفاده می‌کند.

در واقع از آنجایی که نتیجه هر عبارت بولی با جایگذاری مینترم‌هایش، یک می‌شود؛ این متد همه مینترم‌ها را با توجه به تعداد متغیرها که در ورودی از کاربر گرفته می‌شود، در عبارت جایگذاری می‌کند و اگر نتیجه یک شد، true بر می‌گرداند و اگر صفر شد false.

❖ متد foundPI:

این متد بر حسب الگوریتم کویین-مک کلاسکی (تا قبل از جدول پوششی) همه PIها را پیدا می‌کند.

پارامترهای ورودی این متد:

- minterms که یک آرایه دو بعدی از نوع مینترم است که همان مینترم‌های گروه بندی شده بر اساس تعداد یک‌هاست.
- PIs که یک آرایه از نوع مینترم است، لیستی برای ذخیره PIهاست.

این متد یک آرایه دو بعدی جدید از مینترم‌های ترکیب شده را بر می‌گرداند.

الگوریتم:

- هر مینترم را با مینترم‌های گروه بعدی مقایسه می‌کند.
- اگر فاصله شان برابر با یک بود، آنها را ترکیب می‌کند.
- مینترم‌های ترکیب نشده را به عنوان PIهای اولیه ذخیره می‌کند.

❖ متد foundEPI:

این متد PIهای اساسی را با استفاده از جدول پوششی پیدا می‌کند.

پارامترهای این متد:

- PIs یک لیست از نوع مینترم است که همان لیست PIهای اولیه است.
- minterms که آرایه ی مینترم‌های اصلی است.

این متد لیست EPIها را بر می‌گرداند.

مراحل اجرا:

- ساخت جدول پوششی
- یافتن PIهای اساسی (ستون‌هایی با فقط یک ستاره)
- حذف سطرها و ستون‌های پوشش داده شده
- استفاده از منطق petrick برای انتخاب کمترین تعداد PI

در واقع در این متد در مرحله چهارم، برای مجموعه سطرهای باقی مانده (PIهایی که پوشش داده نشده‌اند)، تمام زیر مجموعه‌ها را پیدا می‌کند و چک می‌کند که کدام یک از آنها هم کل سطرهای باقی مانده را پوشش می‌دهد و هم تعداد کمتری از PIها را در خود دارد، سپس همان PIهایی را که در آن زیر مجموعه هستند به عنوان EPI اضافه می‌کند.

نمونه ورودی و خروجی:

به عنوان مثال می‌توان این عبارت را به عنوان ورودی داد:

تعداد متغیرها : 3

عبارت : (A' + B).(C + B')

در این صورت خروجی به این شکل خواهد بود:

A'B' + BC

مثالی دیگر:

تعداد متغیرها : 3

عبارت : (A' + B) (C + B')

خروجی:

invalid expression

نکته مهم: در ورودی دادن دقت کافی را داشته باشید!

And به شکل (.)، Or به شکل (+) و Not به شکل (') است.

اگر n تعداد متغیرها باشد، کاربر فقط مجاز به استفاده از n حرف اول الفبای انگلیسی است!

مستندات فنی:

ساختار پروژه:

- Main.java: کلاس اصلی شامل نقطه ورود برنامه
- Minterm.java: کلاس مینترم برای نگهداری اطلاعات هر مینترم
- Stack.java: پیاده‌سازی ساختار داده پشته

الگوریتم‌های استفاده شده:

- الگوریتم کوین-مک کلاسکی
- روش جدول پوششی
- الگوریتم پشته‌ای برای ارزیابی عبارات بولی (shunting-yard)

جزئیات دقیق الگوریتم:

آماده‌سازی اولیه:

- دریافت و اعتبارسنجی ورودی (با استفاده از متد checkValidation که توضیح داده شد)

```
if(!checkValidation(expression, number)){
    System.out.println("invalid expression");
    return;
}
```

- یافتن مینترم‌های عبارت ورودی

```
String[] array = new String[(int)Math.pow(2, number)];
for(int i = 0; i < array.length; i++) {
    array[i] = intToBinary(i, number);
}
```

در این بخش یک آرایه به نام array از نوع string به طول تعداد مینترم‌های ممکن ساخته شده است و در حلقه با استفاده از متد intToBinary که i (شماره مینترم) را با توجه به تعداد متغیرها (number) تبدیل به باینری می‌کند، مقدار باینری هر مینترم را در خانه با اندیس شماره همان مینترم ذخیره می‌کند.

```
ArrayList<String> mintermsList = new ArrayList<>();
ArrayList<Integer> mintermsNum = new ArrayList<>();
for (int i = 0; i < array.length; i++) {
    if (findMin(expression, array[i])) {
        mintermsList.add(array[i]);
        mintermsNum.add(i);
    }
}
```

در این بخش با استفاده از متد findMin که بالاتر توضیح داده شد، مینترم های عبارت ورودی را به دست آورده و شماره آنها را در mintermsNum و مقدار باینری آنها را در mintermsList ذخیره کرده است.

الگوریتم استفاده شده در متد findMin:

```
public static boolean findMin(String expression, String binary){
    expression = expression.replaceAll(" ", "");
    Stack operandStack = new Stack();
    Stack operatorStack = new Stack();
    char ch, temp;
    for(int i = 0; i < expression.length(); i++){
        ch = expression.charAt(i);
```

در این متد دو شیء از نوع Stack ساخته شده،که operandStack برای عملوند ها و operatorStack برای عملگر ها است. کلاس Stack به صورت ساده پیاده سازی شده است. در حلقه با استفاده از متغیر ch، عبارت پیمایش می شود.

```
// Handle opening parenthesis
if(ch == '('){
    operatorStack.push(ch);
}
// Handle NOT operator
else if(ch == '\''){
    if(operandStack.top() == '1'){
        operandStack.pop();
        operandStack.push('0');
    }
    else{
        operandStack.pop();
        operandStack.push('1');
    }
}
```

در این الگوریتم (shunting-yard) هر زمان به پرانتز باز برسد باید آن را وارد stack کند و هر زمان به پرانتز بسته برسد، باید تمامی عملیات ها را تا رسیدن به اولین پرانتز باز انجام بدهد. هر زمان به Not رسید، باید آخرین عملوند را نقیض کند.

در صفحه بعد کد های مربوط به پرانتز بسته آمده است. در یک حلقه while تا زمانی که به پرانتز باز نرسیده، با استفاده از متد calculateBinary که دو تا عملوند و یک عملگر را به عنوان ورودی گرفته و نتیجه محاسبات را بر می گرداند، عملیات را محاسبه و در آخر آن را با پرانتز باز جایگزین می کند.

در ادامه این کد چک می‌کند که اگر قبل از پرانتز باز And وجود داشت، آن را محاسبه کند.

```
// Handle closing parenthesis
else if(ch == ')'){
    while(operatorStack.top() != '(' && operatorStack.top() > 32){
        temp = operandStack.top();
        operandStack.pop();
        temp = calculateBinary(temp, operatorStack.top(),
                operandStack.top());
        operandStack.pop();
        operatorStack.pop();
        operandStack.push(temp);
    }
    operatorStack.pop();
    if(operatorStack.top() == '.'){
        temp = operandStack.top();
        operandStack.pop();
        temp = calculateBinary(operandStack.top(), '.', temp);
        operandStack.pop();
        operatorStack.pop();
        operandStack.push(temp);
    }
}
```

```
// Handle variables
else if(ch >= 'A' && ch <= 'Z'){
    operandStack.push(binary.charAt(ch - 'A'));
    if((i + 1 < expression.length()) &&
            expression.charAt(i + 1) == '\''){
        if(operandStack.top() == '1'){
            operandStack.pop();
            operandStack.push('0');
        }
        else{
            operandStack.pop();
            operandStack.push('1');
        }
        i++;
    }
    if(operatorStack.top() == '.'){
        temp = operandStack.top();
        operandStack.pop();
        temp = calculateBinary(operandStack.top(), '.', temp);
        operandStack.pop();
        operatorStack.pop();
        operandStack.push(temp);
    }
}
```

این بخش برای زمانی است که به متغیر می‌رسد.

با توجه به اینکه چندمین حرف الفبای انگلیسی است، مقدار آن را از binary می‌گیرد و وارد stack می‌کند. در ادامه‌ی آن وجود عملگر Not را چک می کند؛ سپس با توجه به این که اولویت And بیشتر از Or است، هر موقع به And رسید عملیات را انجام می دهد و بعد از else حلقه پایان می یابد.

```
// Final evaluation of remaining operations
while(!operatorStack.isEmpty()){
    temp = operandStack.top();
    operandStack.pop();
    temp = calculateBinary(operandStack.top(),
            operatorStack.top(), temp);
    operandStack.pop();
    operatorStack.pop();
    operandStack.push(temp);
}
return operandStack.top() == '1';
```

در نهایت تا زمانی که عبارت به صورت کامل محاسبه نشده باشد ادامه می دهد و اگر خروجی یک باشد، متد true بر می گرداند.

- توجه به شرایط خاص

```
// Handle special cases
if(mintermsList.isEmpty()){
    System.out.println("Answer: 0");
    return;
}
if(mintermsNum.size() == Math.pow(2, number)){
    System.out.println("Answer: 1");
    return;
}
```

در این بخش تعداد مینترم‌ها چک می‌شود؛ اگر تعداد آنها صفر بود، یعنی ساده شده‌ی عبارت صفر است و اگر تعداد مینترم‌های عبارت با تعداد مینترم‌های ممکن برابر بود، یعنی ساده شده‌ی عبارت یک است.

```
Minterm[] minterms = new Minterm[mintermsNum.size()];
for (int i = 0; i < mintermsList.size(); i++) {
    minterms[i] = new Minterm();
    minterms[i].setBinary(mintermsList.get(i));
    minterms[i].setNumber(String.valueOf(mintermsNum.get(i)));
}
```

در این قسمت، یک آرایه از نوع Minterm ساخته شده و اطلاعات مینترم ها داخل آن ریخته شده است.

توضیحاتی درباره کلاس Minterm:

در این کلاس دو ویژگی از نوع string وجود دارد که number برای شماره‌ی آن مینترم و binary برای حالت باینری شماره‌ی آن مینترم است. همچنین متدهایی با الگوریتم‌های ساده در این کلاس پیاده‌سازی شده است.

یافتن تمام PIهای اولیه:

- گروه‌بندی مینترم‌ها

```
// Group minterms by number of 1s
int numberOfOnes;
Minterm[][] mintermGroups = new Minterm[number + 1][30];
for (Minterm minterm : minterms) {
    numberOfOnes = minterm.numberOfOnes();
    for (int j = 0; j < 30; j++) {
        if (mintermGroups[numberOfOnes][j] == null) {
            mintermGroups[numberOfOnes][j] = new Minterm();
            mintermGroups[numberOfOnes][j]
                    .setBinary(minterm.getBinary());
            mintermGroups[numberOfOnes][j]
                    .setNumber(minterm.getNumber());
            break;
        }
    }
}
```

در این بخش تعداد یک‌های هر مینترم چک می شود و بر آن اساس در آرایه دو بعدی mintermGroups ذخیره می‌شود. هر شماره سطر در این ماتریس، نشان‌دهنده‌ی تعداد یک‌های مینترم‌های داخل آن سطر است.

- یافتن PIها با استفاده از متد foundPI

```
ArrayList<Minterm> PIList = new ArrayList<>();
Minterm[][] temp = mintermGroups;
for (int i = 0; i < 4; i++) {
    temp = foundPI(temp, PIList);
}
```

در این حلقه ۴ بار گروه‌بندی ها چک می شود و دوباره گروه ایجاد می‌شود (علت ۴ بار اجرا شدن حلقه این است که حداکثر تعداد دفعاتی که نیاز به گروه‌بندی دوباره است، ۴ است)؛ و در ادامه‌ی این کد، PIهای تکراری حذف می شود.

الگوریتم استفاده شده در متد foundPI:

در این متد، آرایه دو بعدی newMintermsCounter برای چک کردن مینترم‌هایی که در گروه‌بندی جدید استفاده نشده اند و آرایه دو بعدی newMinterms برای ذخیره‌سازی گروه‌بندی جدید است. در این متد با استفاده از حلقه‌های تو در تو، هر مینترم با مینترم‌های گروه بعد از خودش مقایسه می شود؛ اگر تفاوت آن مینترم با یکی از مینترم‌های گروه بعد برابر یک بود، با هم ترکیب می‌شوند، تبدیل به یک مینترم می‌شوند، در newMinterms ذخیره می‌شوند و در newMintermsCounter به جای هر دو، ستاره گذاشته می‌شود. اگر به جای یک مینترم، ستاره‌ای در newMintermsCounter نباشد، آن را در لیست PIs اضافه می‌کند.

- یافتن EPIها با استفاده از متد foundEPI

```

ArrayList<Minterm> EPIList = foundEPI(PIs, minterms);

```

الگوریتم این متد بالاتر توضیح داده شد.

چاپ نتیجه:

```
System.out.print("Answer: ");
printAnswer(EPIList);
```