# برنامه نویسی و پیاده سازی الگوریتم کوین-مک‌کلاسکی

روش مینمم سازی توابع بولی

محمد مهدی کرمی

دانشگاه رازی

## چکیده

یک تابع بولی تابعی است که با استفاده از محاسبات منطقی روی مقدار های ورودی بولی آن یک مقدار بولی را بازگشت می‌دهد. این توابع نقش مهمی در برنامه نویسی الگوریتم ها و طراحی مدار های منطقی دارند. کمینه کردن یا مینیمم سازی یک تابع بولی می‌تواند الگوریتم ها و مدارها را بهینه کند. روش کوین-مک‌کلاسکی یکی از تکنیک های قوی برای ساده سازی عبارت های بولی است .روش کوین-مک‌کلاسکی در مقایسه با تکنیک های موجود دیگر قابل اجرا تر است و متغیر های بیشتری را می‌توانند اداره کند. بعلاوه روش کوین- مک‌کلاسکی در پیاده سازی در برنامه های کامپیوتری راحت تر است که آن را به یک روش بهینه تبدیل میکند. این روش از لحاظ تابعی با روش جدول کارنو یکسان است. روش کوین- مک‌کلاسکی می تواند به طور قطعی بیان کند که آیا جواب به حالت کمینه رسیده است.

## معرفی

ساده سازی عبارت های بولی یک ابزار برای بهینه سازی الگوریتم ها و مدار ها است . چندین تکنیک برای مینیمم سازی توابع و عبارات بولی موجود است. یکی از این روش ها روش کوین- مک‌کلاسک (یا به طور خلاصه QMC) می‌باشد. همان طور که در بالا نیز اشاره شد ، روش QMC از نظر پیاده سازی الگوریتم و برنامه نویسی آن ، ساده تر و قابل انجام تر است. چندین روش و الگوریتم برای پیاده سازی روش QM موجود است اما همه ی آن ها ترم های غیر مهم (Don’t care terms) را قبول و پشتیبانی نمی‌کنند یا محدودیت هایی با تعداد متغیر های ورودی دارند. در اینجا کد شبیه سازی روش QM را بر پایه زبان پایتون معرفی می‌کنیم. این کد و الگوریتم محدودیت ای در تعداد متغیر های ورودی ندارد و ترم های غیر مهم را نیز قبول و پشتیبانی می‌کند.

## نحوه‌ی عمل و پیاده سازی الگوریتم QMC

### معرفی نحوه‌ی عملکرد الگوریتم

روش QMC بر اساس قواعد ساده سازی جبر بولی است ، بصورتی که

$$AB+AB'=A$$

در این فرمول، A میتواند متغیر یا مجموعه ای متغیر ها باشد ، و B نیز یک متغیر باشد. این به این معنی است هنگامی که همه ی متغیر های دو ترم بجز فقط یک متغیر برابر باشد، می‌توان این دو متغیر را با هم ترکیب کرد و یک ترم جدید با یک لیترال (literal) کمتر بوجود آورد. همه ی ترم ها در تابع بولی برای ترکیب احتمالی هر دو ترم باهم تست می‌شوند و یک مجموعه ی جدید شامل ترم های جدیدی که یک لیترال کمتر دارند تشکیل می‌شود و همین پروسه و روند روی مجموعه ی جدید ترم ها تا جایی تکرار می‌شود که هیچکدام از ترم های موجود در مجموعه را نتوان با یکدیگر ترکیب کرد.

ترم هایی که تفکیک ناپذیر هستند (نمیتوانند با ترم های دیگر ترکیب شوند) را "ایجاب کننده های اول" ( implicants prime یا به طور خالصه PI ) می‌نامند. قدم نهایی انتخاب PI هایی است که شامل کمترین تعداد PI هایی باشند که میتوان با آن ها تمامی ترم های ورودی اصلی (مینترم های ورودی) را پوشش داد. PI های انتخاب شده را "ایجاب کنند های اول اصلی(implicants prime Essential یا به طور خالصه EPI ) می‌نامند. EPIها همان حالت کمینه یا مینیمم شده عبارت هستند.

## نحوه عملکرد متود QM و الگوریتم برای پیاده سازی آن

ورودی های الگوریتم:

- تابع F به صورت نرمال ترکیب فصلی اساسی
- در صورت وجود مینترم های غیر مهم (Don’t care Minterms)
- تعداد متغیر های تابع

مرحله اول

مینترم های تابع F به باینری تبدیل می‌شوند و ترم ها به وجود می‌آیند.

هر ترم با i بیت ’1‘ در گروه i ام ذخیره می‌شوند.

ستون اول بوجود می‌آید.

مرحله دوم

ترم های هر گروه را با ترم های گروه بعدی آنان مقایسه می‌شوند. در صورتی که تمامی بیت ها بجز یک بیت در هر دو ترم باهم در یک مکان ثابت برابر بود ، این دو ترم قابل ترکیب شدن هستن.

اگر دو ترم x و y از گروه‌های i ام و i+1 ام باهم قابل ترکیب شدن بودن ، بیت جدید که ترم x یا y است به طوری که در مکانی که بیت متفاوت موجود بود ، ’_‘ را قرار داده شده و ترم جدید را در گروه i ام ستون بعدی همراه با مینترم های که این ترم حاصل از ترکیب شدن آن ها بوده ذخیره می‌شود.

ترم x و y به عنوان ترم های ترکیب شده علامت گذاری می‌شوند.

مرحله سوم

مرحله ی دوم تا جایی برای ستون های بعدی تکرار می‌شود که دیگر هیچ ترمی قابل ترکیب شدن موجود نباشد.

مرحله چهارم

تمامی ترم های ترکیب نشده یا علامت گذاری نشده ، PI ها هستند. تمامی PI ها را در جدول ایجاب کننده های اول یا PI chart قرار داده. در این جدول مشخص می‌شود که هر ترم از ترکیب چه مینترم تشکیل داده شده‌اند.

منترم هایی که فقط توسط یک ترم پوشش داده شده باشند اولین سری از EPI ها یا ایجاب کننده ای اول اساسی را تشکیل می-دهند.

این EPI ها و مینترم‌های تشکیل دهنده‌ی آن ها از جدول PI chart حذف شده و جدول Reduced PI chart بوجود می‌آید

مرحله پنجم

جدول Reduced PI chart در صورت خالی نبودن حاوی EPI های باقیمانده است.

کمترین تعداد PI که تمامی مینترم های باقیمانده را پوشش دهند ، EPI های باقیمانده را بوجود می‌آورند.

مرحله ششم

EPI ها تبدیل به صورت نرمال ترکیب فصلی می‌شوند و صورت مینیمم شده بدست می‌آید.

## عملکرد برنامه QMC در پایتون

برای توضیح دقیق تر نحوه عملکرد برنامه ، متغیر های زیر را برای ورودی برنامه در نظر بگیرید:

$$F(A,B,C,D) = \sum m(4,5,6,9,11,12,13,14) + \sum d(0,1,3,7)$$

که حاوی در کل ۱۲ مینترم شامل ۴ ترم غیر مهم (Don’t care Minterms) می‌باشد.

تعداد متغیر های ورودی ۴ می‌باشد.

در قدم اول تمامی مینترم ها و ترم های غیر مهم به باینری تبدیل شده و در مجموعه decMinterms ذخیره می‌شوند.

تمامی ترم ها با i بیت ۱ در decMinterms با ساختمان داده زیر در گروه i ام ستون صفرم ذخیره می‌شود. (col[0][i])

| Term | incDec | Paired(0) | Xpos: [] |
|---|---|---|---|

که incDec لیست تمامی مینترم هایی می‌باشد که Term از ترکیب آنان تشکیل شده است ( در ستون صفرم همان مینترم بوجود آورنده ترم یا معادل دسیمال آن می‌باشد)

Paired نشان دهنده این است که آیا این ترم با ترم های دیگه ترکیب شده است یا نه ، که اگر ترکیب شده باشد برابر یک ، در غیر اینصورت برابر صفر میشود. در ابتدا برابر صفر قرار داده می‌شود و در صورت ترکیب شدن ترم با ترم دیگر برابر ۱ می‌شود.

Xpos یک لیست است که نشان دهنده ی محل قرار گیری بیت های حذف شده (بیت هایی که حذف میشوند ، x نامیده میشوند) با توجه به این که هنوز ترکیب ترم ای صورت نگرفته ، برابر یک لیست خالی است.

پس در این مرحله ستون صفرم به شکل زیر در خواهد آمد

| | Term | incDec | paired | Xpos |
|---|---|---|---|---|
| Group 0 | 0000 | [0] | 0 | [ ] |
| Group 1 | 0001 | [1] | 0 | [ ] |
| | 0100 | [4] | 0 | [ ] |
| Group 2 | 0011 | [3] | 0 | [ ] |
| | 0101 | [5] | 0 | [ ] |
| | 0110 | [6] | 0 | [ ] |
| | 1001 | [9] | 0 | [ ] |
| | 1100 | [12] | 0 | [ ] |
| Group 3 | 0111 | [7] | 0 | [ ] |
| | 1011 | [11] | 0 | [ ] |
| | 1101 | [13] | 0 | [ ] |
| | 1110 | [14] | 0 | [ ] |

سپس تمامی ترم های هر گروه با گروه بعدی با توجع به قواعد QMC مقایسه می‌شوند

در صورتی که هر دو ترم با هم ترکیب شدند ، Paired در این دو ترم برابر با یک شده و ترم جدید در ستون بعدی ذخیره خواهد شد.

همین روند را برای ستون های بعدی تا جایی ادامه می‌دهیم که هیچ ترم جدیدی قابلیت ترکیب شدند نداشته باشند.

پس از ترکیب ترم ها خواهیم داشت:

col[0]:

| | Term | incDec | paired | Xpos |
|---|---|---|---|---|
| Group 0 | 0000 | [0] | 1 | [ ] |
| Group 1 | 0001 | [1] | 1 | [ ] |
| | 0100 | [4] | 1 | [ ] |
| Group 2 | 0011 | [3] | 1 | [ ] |
| | 0101 | [5] | 1 | [ ] |
| | 0110 | [6] | 1 | [ ] |
| | 1001 | [9] | 1 | [ ] |
| | 1100 | [12] | 1 | [ ] |
| Group 3 | 0111 | [7] | 1 | [ ] |
| | 1011 | [11] | 1 | [ ] |
| | 1101 | [13] | 1 | [ ] |
| | 1110 | [14] | 1 | [ ] |

col[1]:

| | Term | incDec | paired | Xpos |
|---|---|---|---|---|
| Group 0 | 000_ | [0, 1] | 1 | [3] |
| | 0_00 | [0, 4] | 1 | [1] |
| Group 1 | 00_1 | [1,3] | 1 | [2] |
| | 0_01 | [1,5] | 1 | [1] |
| | _001 | [1,9] | 1 | [0] |
| | 010_ | [4,5] | 1 | [3] |
| | 01_0 | [4,6] | 1 | [2] |
| | _100 | [4,12] | 1 | [0] |
| Group 2 | 0_11 | [3, 7] | 1 | [1] |
| | _011 | [3, 11] | 1 | [0] |
| | 01_1 | [5, 7] | 1 | [2] |
| | _101 | [5, 13] | 1 | [0] |
| | 011_ | [6, 7] | 1 | [3] |
| | _110 | [6, 14] | 1 | [0] |
| | 10_1 | [9, 11] | 1 | [2] |
| | 1_01 | [9, 13] | 1 | [1] |
| | 110_ | [12, 13] | 1 | [3] |
| | 11_0 | [12, 14] | 1 | [2] |

col[2]:

| | Term | incDec | paired | Xpos |
|---|---|---|---|---|
| Group 0 | 0_0_ | [0, 1, 4, 5] | 0 | [3, 1] |
| Group 1 | 0__1 | [1, 3, 5, 7] | 0 | [2,1] |
| | _0_1 | [1, 3, 9, 11] | 0 | [2, 0] |
| | __01 | [1, 5, 9, 13] | 0 | [1, 0] |
| | 01__ | [4, 5, 6, 7] | 0 | [3, 2] |
| | _10_ | [4, 5, 12, 13] | 0 | [3, 0] |
| | _1_0 | [4, 6, 12, 14] | 0 | [2, 0] |

پس از انجام این مراحل ، چون هیچ ترم دیگری قابلیت ترکیب شدن با ترم دیگری را ندارد ، PI ها با شرط paired = 0 ، جدول ایجاب کننده های اول یا PI chart تشکیل داده می‌دهند.

برای اتلاف فضای ذخیره سازی و جلوگیری از تخریب اطلاعات جدول اصلی یا col فقط آدرس PI ها در col را در PI chart ذخیره می‌شود.

همچنین در این مرحله لیست NumberCounter نیز تشکیل داده می‌شود ، این لیست نشان دهنده ی تعداد دفعاتی است که هر مینترم در یک PI ظاهر می‌شود. به عنوان مثال اگر مینترم 4 در سه PI ظاهر شده باشد NumberCounter[4] برابر ۳ می‌شود.

در این مرحله برای ورودی ها خواهیم داشت

PI_chart_index:

[[2, 0, 0], [2, 1, 0], [2, 1, 1], [2, 1, 2], [2, 1, 3], [2, 1, 4], [2, 1, 5]]

این لیست PI_chart_index همانطور که گفته شد آدرس PI ها در col میباشد ، یعنی [2,0,0] به ترمی که در ستون دوم ، گروه صفرم و ترمی که در مکان صفرم این گروه وجود دارد اشاره می‌کند.

اگر این آدرس ها را به عبارتی ترجمه شوند PI ها را می‌توان مشاهده نمود.

['0_0_', [0, 1, 4, 5], 0, [3, 1]]

['0__1', [1, 3, 5, 7], 0, [2, 1]]

['_0_1', [1, 3, 9, 11], 0, [2, 0]]

['__01', [1, 5, 9, 13], 0, [1, 0]]

['01__', [4, 5, 6, 7], 0, [3, 2]]

['_10_', [4, 5, 12, 13], 0, [3, 0]]

['_1_0', [4, 6, 12, 14], 0, [2, 0]]

و برای NumberCounter خواهیم داشت:

[1, 4, 0, 2, 4, 5, 2, 2, 0, 2, 0, 1, 2, 2, 1, 0]

دو لیست NumberCounter و PI_chart_index در برنامه نمایانگر جدول زیر که همان جدول PI chart در برنامه می‌باشند.

| | | 0 | 1 | 3 | 4 | 5 | 6 | 7 | 9 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0, 1, 4, 5 | 0_0_ | O | O | | O | O | | | | | | | |
| 1, 3, 5, 7 | 0__1 | | O | O | | O | | O | | | | | |
| 1, 3, 9, 11 | _0_1 | | O | O | | | | | O | O | | | |
| 1, 5, 9, 13 | __01 | | O | | | O | | | O | | | O | |
| 4, 5, 6, 7 | 01__ | | | | O | O | O | O | | | | | |
| 4, 5, 12, 13 | _10_ | | | | O | O | | | | | O | O | |
| 4, 6, 12, 14 | _1_0 | | | | O | | O | | | | O | | O |

تمامی مینترم هایی که فقط توسط یک PI پوشش داده شده اند ، مینترم های اساسی هستند و این PI ها EPI ها را تشکیل می‌دهند.

همانطور که مشاهده می‌شود دور ستون مینترم های اساسی با رنگ سبز خط کشیده شده است. دور EPI ها نیز با رنگ آبی خط کشیده شده است.

ستون مینترم های غیرمهم نیز حذف می‌شوند چون در محاسبه EPI ها تاثیری نخواهند داشت ، دور این ستون ها نیز با خط قرمز کشیده شده است.

تمامی مینترم هایی که توسط EPI ها پوشش داده شده اند از این جدول حذف خواهد شد (NumberCounter[i]=0 )

آدرس EPI ها نیز از PI_chart_index حذف خواهند شد.

پس از این اعمال جدول Reduced PI chart از جدول chart PI ساخته می‌شود.

| | | 5 | 13 |
|---|---|---|---|
| 0, 1, 4, 5 | 0_0_ | O | |
| 1, 3, 5, 7 | 0__1 | O | |
| 1, 5, 9, 13 | __01 | O | O |
| 4, 5, 6, 7 | 01__ | O | |
| 4, 5, 12, 13 | _10_ | O | O |

در این جدول برنامه سعی میکند بیشترین تعداد مینترم را توسط کمترین تعداد از PI ها را پوشش دهد.

چون اولین آدرس در PI_chart_index که بیشترین پوشش مینترم ها را دارد 1__0 می‌باشد ، برنامه این ترم را انتخاب کرده و در نهایت تمامی EPI ها به صورت نرمال ترکیب فصلی ترکیب شده و منیمم تابع ورودی بدست خواهد آمد.

$$F(A,B,C,D) = B'D+BD'+C'D$$

علاوه بر این جواب چون در جدول PI_chart_index یک PI دیگر نیز قابلیت EPI بودن را دارد میتوان عبارت BC' را بجای عبارت C'D قرار داد و یک منیمم دیگر داشت.

$$F(A,B,C,D) = B'D+BD'+BC'$$

## کد الگوریتم QMC در پایتون

```
def decimalToBinary(n,varCount):
    #convert input decimal of n into binary

    t = bin(n).replace("0b", "")
    if len(t)<varCount:
        t= '0'*(varCount - len(t)) + t
    return t

def grpPrep(varCount):
    #prepares a proper structure of groups for saving new terms
    g=list()
    for i in range(varCount+1):
        g.append([])
    return g
```

```
def digitCombine(x,y,varCount,c):
    #combines input bit of x , y and c is the of Xs

    counter= 0
    newBit=''
    w=list(c)
    for index in range(0,varCount):
        if x[index] == y[index]:
            newBit+=x[index]
        else:
            newBit+='_'
            w+=[index]
            counter+=1
    if counter>1:
        return False
    else:
        return [newBit,w]
        #w is the position of the new X


def checkDup(grp,newTerm):
    #check if there are any terms in grp as same as newTerm
    for term in grp:
        if term[0]==newTerm:
            return 0
    return 1


def pairing(col,varCount,i=0):
    col.append([])
    col[i+1]=(grpPrep(varCount))

    anyPairing = 0
    #gc: group counter
    for gc in range(varCount):

        #check if the group gc is not empty
        if col[i][gc]:

            #pick terms (like termA) from group gc to compare with-
            # -terms (like termsB) in next group (gc+1)
            for termA in col[i][gc]:

                #pick terms(like termB) from group (gc+1) to compare with
(termA)
                for termB in col[i][gc+1]:

                    #check if Xposes matches in termA and termB
                    if termA[3]==termB[3]:

                        #combine termA and termB if possible
```

```
                        newTermAndXpos =
digitCombine(termA[0],termB[0],varCount,termA[3] )

                        #check if combining termA and termB was possible
                        if newTermAndXpos:

                            #mark termA and termB as paired
                            termA[2]=termB[2]=1

                            #check if there are any term same as the new term
in the next column(i+1)
                            if
checkDup(col[i+1][newTermAndXpos[0].count("1")],newTermAndXpos[0]):

                                #determine incDec for the new term and sort
it
                                incDec=termA[1]+termB[1]
                                incDec.sort()

                                #append the new term with proper data
structure-
                                #-into the correct
group(newTermAndXpos[0].count("1")) in the next column (i+1)

col[i+1][newTermAndXpos[0].count("1")].append([newTermAndXpos[0],incDec,0,new
TermAndXpos[1]])

                                anyPairing=1
    if anyPairing:
        pairing(col,varCount,i+1)


def piChart(col,varCount):
    #creating piChart

    PI_chart_index = list()
    #Creating NumberCounter structure
    NumberCounter=[0]*2**varCount

    for cl in col:
        for grp in cl:
            #if group is not empty
            if grp:
                for term in grp:

                    #if term is not paired
                    if not term[2]:
                        #adding address of not paired term to PI_chart_index

PI_chart_index.append([col.index(cl),cl.index(grp),grp.index(term)])
                        for incDec in term[1]:
                            #completing NumberCounter
```

```
                                NumberCounter[incDec]+=1

    return (PI_chart_index,NumberCounter)


def findEPI(col, PI_chart_index , NumberCounter ,dontCareMint):
    #finding EPIs

    EPI_index=[]
    EPI_dec=[]

    #if there exist input dontCareMint
    if dontCareMint:
        for i in dontCareMint:
            #put 0 in their place on NumberCounter
            NumberCounter[i]=0

    for i in range(len(NumberCounter)):
        #finding minterms that can only be covered by 1 term
        if NumberCounter[i] == 1:
            EPI_dec.append(i)


    for cl in col:
        for grp in cl:
            if grp:
                for term in grp:
                    if not term[2]:
                        if any(True for x in EPI_dec if x in term[1]):
                            adr=[col.index(cl),cl.index(grp),
grp.index(term)]
                            EPI_index.append(adr)
                            PI_chart_index.remove(adr)
                            for incDec in term[1]:
                                NumberCounter[incDec]=0

    return EPI_index


def reducedPI(col,PI_chart_index,remaningMinterms,EPI_index,step=0):

    #finding the rest of the EPIs or basically reduced PI chart

    #safe check for when an EPI is discovered  and the program would move on
    success=0

    #satisfyingLen is the number of minterms that we want to cover by 1 PI

    if len(remaningMinterms)>1:
        satisfyingLen= len(remaningMinterms) - step
    else:
        satisfyingLen = 1
```

```
    for adr in PI_chart_index :
        if not success:
            counter = 0
            incDecOfEPI=[]
            for rm in remaningMinterms:
                #rm : one of the Remaning Minterms
                if rm in col[adr[0]][adr[1]][adr[2]][1]:
                    incDecOfEPI.append(rm)
                    counter+=1

            if counter == satisfyingLen:
                #success=1
                EPI_index.append(adr)
                PI_chart_index.remove(adr)
                for i in incDecOfEPI:
                    remaningMinterms.remove(i)
    if remaningMinterms :
        if not success:
            reducedPI(col,PI_chart_index,remaningMinterms,EPI_index,step+1)


def cycle(x,varCount,c=''):
    al= list("ABCDEFG")
    p = varCount - len(x)
    if len(x) != 0 :
        if(x[0] == '0'):
            c += al[p]+"'"
        if(x[0] == '1'):
            c += al[p]
    if len(x)>1:
        return cycle(x[1:],varCount,c)
    if len(x)==1:
        return c

def toLiterals(col,EPI_index,varCount):
    minli=[]
    li=[]
    for adr in EPI_index:
        li.append(col[adr[0]][adr[1]][adr[2]][0])

    for term in li:
        if term == '0'*varCount :
            minli.append("A'B'C'D'E'F'G'"[0:2*varCount])
        else:
            c= list(term)
            #print(c)
            minli.append(cycle(c,varCount))
    print(f"minimized SOP by literals = ", ' + '.join(minli))
```

In [25]:

```
def printCol(col):
    i=0
```

```
    for cl in col:
        g=0
        print(f"** coloumn {i}")
        i+=1
        for grp in cl:
            if grp:
                print(f"*** group {g}")
                g+=1
                for term in grp:
                    print(f"\t- {term}")

def printAdr(col,adrs):
    i=0
    for adr in adrs:
        print(f"{i} - {col[adr[0]][adr[1]][adr[2]]}")
        i+=1



def QMC(minterms , dontCareMint=[],varCount=4):

    col=[[],[]]
    col[0]=(grpPrep(varCount))



    decMinterms=
set(list(map(int,minterms.strip().split()))).union(set(dontCareMint))


    for minterm in decMinterms:

col[0][bin(minterm).count("1")].append([decimalToBinary(minterm,varCount),[mi
nterm], 0,[]])
        # data structure [ term , incDec , paired , Xpos]
    pairing(col,varCount)


    #print(f"\t x entering minterms into the 0 column")
    #printCol(col)


    PI_chart_index , NumberCounter=piChart(col,varCount)

    #printCol(col)

    #print(f"\n\t x PI chart step x")
    #printAdr(col,PI_chart_index)


    #print(f"\n\tPI_chart_index\n")
    #print(f"PI_chart_index: {PI_chart_index}")
```

```
    #print(f"NumberCounter: {NumberCounter}")



    EPI_index = findEPI(col, PI_chart_index , NumberCounter ,dontCareMint)


    #print(f"\n\t x EPI chart step x")
    #printAdr(col,EPI_index)
    #print(f"\n\t  Reduced PI chart step ")
    #printAdr(col,PI_chart_index)



    remaningMinterms=[]
    nc=0
    for index in NumberCounter:
        if index:
            remaningMinterms.append(nc)
        nc+=1

    if remaningMinterms:
        reducedPI(col,PI_chart_index,remaningMinterms,EPI_index)

    #print(f"\n\t x All of the EPIs after reducedPI ")
    #printAdr(col,EPI_index)

    toLiterals(col,EPI_index,varCount)



def main():
    print(f"* Welcome to Quine-McCluskey minimization method program *\n")
    varCount=int(input("\nEnter the number of variables : \n"))
    Minterms= input("\nEnter the minterms : ")
    x=input("\nIs there any Don't care minterms in the function ? (y/n)\t")
    if x=='y':
        dontCareMint=list(map(int,input("\nEnter Don't care
terms:").strip().split()))
    else:
        dontCareMint=[]

    QMC(Minterms , dontCareMint ,varCount)

    c=int(input("\nEnter 1 to start again , 0 to exit the program\n"))

    if c==1:
        main()


main()
```