```
# Author: Gagandeep Singh 112009011
Quine-McCluskey method for Max ternms and Min terms

press =int(input('Enter 1 for maxterms or press 2 for min terms :'))
if press == 1:
   def mul(x,y): # Multiply 2 minterms
    res = []
    for i in x:
      if i+"'" in y or (len(i)==2 and i[0] in y):
         return []
      else:
         res.append(i)
    for i in y:
      if i not in res:
         res.append(i)
    return res

   def multiply(x,y): # Multiply 2 expressions
      res = []
      for i in x:
         for j in y:
            tmp = mul(i,j)
            res.append(tmp) if len(tmp) != 0 else None
      return res

   def refine(my_list,dc_list): # Removes don't care terms from a given list and returns refined list
      res = []
      for i in my_list:
         if int(i) not in dc_list:
            res.append(i)
      return res

   def findEPI(x): # Function to find essential prime implicants from prime implicants chart
      res = []
      for i in x:
         if len(x[i]) == 1:
            res.append(x[i][0]) if x[i][0] not in res else None
      return res

   def findVariables(x): # Function to find variables in a minterm. For example, the minterm --01 has C
and D' as variables
      var_list = []
      for i in range(len(x)):
         if x[i] == '0':
            var_list.append(chr(i+65))
         elif x[i] == '1':
            var_list.append(chr(i+65)+"'")
      return var_list

   def flatten(x): # Flattens a list
      flattened_items = []
      for i in x:
         flattened_items.extend(x[i])
      return flattened_items
```

```
    def findminterms(a): #Function for finding out which minterms are merged. For example, 10-1 is
obtained by merging 9(1001) and 11(1011)
        gaps = a.count('-')
        if gaps == 0:
            return [str(int(a,2))]
        x = [bin(i)[2:].zfill(gaps) for i in range(pow(2,gaps))]
        temp = []
        for i in range(pow(2,gaps)):
            temp2,ind = a[:],-1
            for j in x[0]:
                if ind != -1:
                    ind = ind+temp2[ind+1:].find('-')+1
                else:
                    ind = temp2[ind+1:].find('-')
                temp2 = temp2[:ind]+j+temp2[ind+1:]
            temp.append(str(int(temp2,2)))
            x.pop(0)
        return temp

    def compare(a,b): # Function for checking if 2 minterms differ by 1 bit only
        c = 0
        for i in range(len(a)):
            if a[i] != b[i]:
                mismatch_index = i
                c += 1
                if c>1:
                    return (False,None)
        return (True,mismatch_index)

    def removeTerms(_chart,terms): # Removes minterms which are already covered from chart
        for i in terms:
            for j in findminterms(i):
                try:
                    del _chart[j]
                except KeyError:
                    pass

    mt = [int(i) for i in input("Enter the maxterms: ").strip().split()]
    dc = [int(i) for i in input("Enter the don't cares(If any): ").strip().split()]
    mt.sort()
    minterms = mt+dc
    minterms.sort()
    size = len(bin(minterms[-1]))-2
    groups,all_pi = {},set()

    # Primary grouping starts
    for minterm in minterms:
        try:
            groups[bin(minterm).count('1')].append(bin(minterm)[2:].zfill(size))
        except KeyError:
            groups[bin(minterm).count('1')] = [bin(minterm)[2:].zfill(size)]
    # Primary grouping ends

    #Primary group printing starts
    print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
    for i in sorted(groups.keys()):
        print("%5d:"%i) # Prints group number
```

```
      for j in groups[i]:
         print("\t\t    %-20d%s"%(int(j,2),j)) # Prints minterm and its binary representation
      print('-'*50)
   #Primary group printing ends

   # Process for creating tables and finding prime implicants starts
   while True:
      tmp = groups.copy()
      groups,m,marked,should_stop = {},0,set(),True
      l = sorted(list(tmp.keys()))
      for i in range(len(l)-1):
         for j in tmp[l[i]]: # Loop which iterates through current group elements
            for k in tmp[l[i+1]]: # Loop which iterates through next group elements
               res = compare(j,k) # Compare the minterms
               if res[0]: # If the minterms differ by 1 bit only
                  try:
                     groups[m].append(j[:res[1]]+'-'+j[res[1]+1:]) if j[:res[1]]+'-'+j[res[1]+1:] not in
groups[m] else None # Put a '-' in the changing bit and add it to corresponding group
                  except KeyError:
                     groups[m] = [j[:res[1]]+'-'+j[res[1]+1:]] # If the group doesn't exist, create the group at
first and then put a '-' in the changing bit and add it to the newly created group
                  should_stop = False
                  marked.add(j) # Mark element j
                  marked.add(k) # Mark element k
         m += 1
      local_unmarked = set(flatten(tmp)).difference(marked) # Unmarked elements of each table
      all_pi = all_pi.union(local_unmarked) # Adding Prime Implicants to global list
      print("Unmarked elements(Prime Implicants) of this table:",None if len(local_unmarked)==0 else ',
'.join(local_unmarked)) # Printing Prime Implicants of current table
      if should_stop: # If the minterms cannot be combined further
         print("\n\nAll Prime Implicants: ",None if len(all_pi)==0 else ', '.join(all_pi)) # Print all prime
implicants
         break
      # Printing of all the next groups starts
      print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
      for i in sorted(groups.keys()):
         print("%5d:"%i) # Prints group number
         for j in groups[i]:
            print("\t\t%-24s%s"%(','.join(findminterms(j)),j)) # Prints minterms and its binary
representation
         print('-'*50)
      # Printing of all the next groups ends
   # Process for creating tables and finding prime implicants ends


   # Printing and processing of Prime Implicant chart starts
   sz = len(str(mt[-1])) # The number of digits of the largest minterm
   chart = {}
   print('\n\n\nPrime Implicants chart:\n\n    Minterms    |%s\n%s'%(' '.join((' '*(sz-len(str(i))))+str(i)
for i in mt),'='*(len(mt)*(sz+1)+16)))
   for i in all_pi:
      merged_minterms,y = findminterms(i),0
      print("%-16s|"%','.join(merged_minterms),end='')
      for j in refine(merged_minterms,dc):
         x = mt.index(int(j))*(sz+1) # The position where we should put 'X'
         print(' '*abs(x-y)+' '*(sz-1)+'X',end='')
         y = x+sz
```

```
            try:
                chart[j].append(i) if i not in chart[j] else None # Add minterm in chart
            except KeyError:
                chart[j] = [i]
        print('\n'+'-'*(len(mt)*(sz+1)+16))
    # Printing and processing of Prime Implicant chart ends

    EPI = findEPI(chart) # Finding essential prime implicants
    print("\nEssential Prime Implicants: "+', '.join(str(i) for i in EPI))
    removeTerms(chart,EPI) # Remove EPI related columns from chart

    if(len(chart) == 0): # If no minterms remain after removing EPI related columns
        final_result = [findVariables(i) for i in EPI] # Final result with only EPIs
    else: # Else follow Petrick's method for further simplification
        P = [[findVariables(j) for j in chart[i]] for i in chart]
        while len(P)>1: # Keep multiplying until we get the POS form of P
            P[1] = multiply(P[0],P[1])
            P.pop(0)
        final_result = [min(P[0],key=len)] # Choosing the term with minimum variables from P
        final_result.extend(findVariables(i) for i in EPI) # Adding the EPIs to final solution
    print('\n\nSolution: Expression(max) = '+' * '.join('+'.join(i) for i in final_result))

    input("\nPress enter to exit...")
elif press == 2:
    def mul(x,y): # Multiply 2 minterms
     res = []
     for i in x:
        if i+"'" in y or (len(i)==2 and i[0] in y):
            return []
        else:
            res.append(i)
     for i in y:
        if i not in res:
            res.append(i)
     return res

    def multiply(x,y): # Multiply 2 expressions
        res = []
        for i in x:
            for j in y:
                tmp = mul(i,j)
                res.append(tmp) if len(tmp) != 0 else None
        return res

    def refine(my_list,dc_list): # Removes don't care terms from a given list and returns refined list
        res = []
        for i in my_list:
            if int(i) not in dc_list:
                res.append(i)
        return res

    def findEPI(x): # Function to find essential prime implicants from prime implicants chart
        res = []
        for i in x:
            if len(x[i]) == 1:
                res.append(x[i][0]) if x[i][0] not in res else None
        return res
```

```
   def findVariables(x): # Function to find variables in a minterm. For example, the minterm --01 has C and D' as variables
      var_list = []
      for i in range(len(x)):
         if x[i] == '0':
            var_list.append(chr(i+65)+"'")
         elif x[i] == '1':
            var_list.append(chr(i+65))
      return var_list

   def flatten(x): # Flattens a list
      flattened_items = []
      for i in x:
         flattened_items.extend(x[i])
      return flattened_items

   def findminterms(a): #Function for finding out which minterms are merged. For example, 10-1 is obtained by merging 9(1001) and 11(1011)
      gaps = a.count('-')
      if gaps == 0:
         return [str(int(a,2))]
      x = [bin(i)[2:].zfill(gaps) for i in range(pow(2,gaps))]
      temp = []
      for i in range(pow(2,gaps)):
         temp2,ind = a[:],-1
         for j in x[0]:
            if ind != -1:
               ind = ind+temp2[ind+1:].find('-')+1
            else:
               ind = temp2[ind+1:].find('-')
            temp2 = temp2[:ind]+j+temp2[ind+1:]
         temp.append(str(int(temp2,2)))
         x.pop(0)
      return temp

   def compare(a,b): # Function for checking if 2 minterms differ by 1 bit only
      c = 0
      for i in range(len(a)):
         if a[i] != b[i]:
            mismatch_index = i
            c += 1
            if c>1:
               return (False,None)
      return (True,mismatch_index)

   def removeTerms(_chart,terms): # Removes minterms which are already covered from chart
      for i in terms:
         for j in findminterms(i):
            try:
               del _chart[j]
            except KeyError:
               pass

   mt = [int(i) for i in input("Enter the minterms: ").strip().split()]
   dc = [int(i) for i in input("Enter the don't cares(If any): ").strip().split()]
   mt.sort()
```

```
    minterms = mt+dc
    minterms.sort()
    size = len(bin(minterms[-1]))-2
    groups,all_pi = {},set()

    # Primary grouping starts
    for minterm in minterms:
        try:
            groups[bin(minterm).count('1')].append(bin(minterm)[2:].zfill(size))
        except KeyError:
            groups[bin(minterm).count('1')] = [bin(minterm)[2:].zfill(size)]
    # Primary grouping ends

    #Primary group printing starts
    print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
    for i in sorted(groups.keys()):
        print("%5d:"%i) # Prints group number
        for j in groups[i]:
            print("\t\t    %-20d%s"%(int(j,2),j)) # Prints minterm and its binary representation
        print('-'*50)
    #Primary group printing ends

    # Process for creating tables and finding prime implicants starts
    while True:
        tmp = groups.copy()
        groups,m,marked,should_stop = {},0,set(),True
        l = sorted(list(tmp.keys()))
        for i in range(len(l)-1):
            for j in tmp[l[i]]: # Loop which iterates through current group elements
                for k in tmp[l[i+1]]: # Loop which iterates through next group elements
                    res = compare(j,k) # Compare the minterms
                    if res[0]: # If the minterms differ by 1 bit only
                        try:
                            groups[m].append(j[:res[1]]+'-'+j[res[1]+1:]) if j[:res[1]]+'-'+j[res[1]+1:] not in
groups[m] else None # Put a '-' in the changing bit and add it to corresponding group
                        except KeyError:
                            groups[m] = [j[:res[1]]+'-'+j[res[1]+1:]] # If the group doesn't exist, create the group at
first and then put a '-' in the changing bit and add it to the newly created group
                        should_stop = False
                        marked.add(j) # Mark element j
                        marked.add(k) # Mark element k
            m += 1
        local_unmarked = set(flatten(tmp)).difference(marked) # Unmarked elements of each table
        all_pi = all_pi.union(local_unmarked) # Adding Prime Implicants to global list
        print("Unmarked elements(Prime Implicants) of this table:",None if len(local_unmarked)==0 else ',
'.join(local_unmarked)) # Printing Prime Implicants of current table
        if should_stop: # If the minterms cannot be combined further
            print("\n\nAll Prime Implicants: ",None if len(all_pi)==0 else ', '.join(all_pi)) # Print all prime
implicants
            break
        # Printing of all the next groups starts
        print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='*50))
        for i in sorted(groups.keys()):
            print("%5d:"%i) # Prints group number
            for j in groups[i]:
                print("\t\t%-24s%s"%(','.join(findminterms(j)),j)) # Prints minterms and its binary
representation
```

```
            print('-'*50)
        # Printing of all the next groups ends
    # Process for creating tables and finding prime implicants ends


    # Printing and processing of Prime Implicant chart starts
    sz = len(str(mt[-1])) # The number of digits of the largest minterm
    chart = {}
    print('\n\n\nPrime Implicants chart:\n\n    Minterms    |%s\n%s'%(' '.join((' '*(sz-len(str(i))))+str(i)
for i in mt),'='*(len(mt)*(sz+1)+16)))
    for i in all_pi:
        merged_minterms,y = findminterms(i),0
        print("%-16s|"%','.join(merged_minterms),end='')
        for j in refine(merged_minterms,dc):
            x = mt.index(int(j))*(sz+1) # The position where we should put 'X'
            print(' '*abs(x-y)+' '*(sz-1)+'X',end='')
            y = x+sz
            try:
                chart[j].append(i) if i not in chart[j] else None # Add minterm in chart
            except KeyError:
                chart[j] = [i]
        print('\n'+'-'*(len(mt)*(sz+1)+16))
    # Printing and processing of Prime Implicant chart ends

    EPI = findEPI(chart) # Finding essential prime implicants
    print("\nEssential Prime Implicants: "+', '.join(str(i) for i in EPI))
    removeTerms(chart,EPI) # Remove EPI related columns from chart

    if(len(chart) == 0): # If no minterms remain after removing EPI related columns
        final_result = [findVariables(i) for i in EPI] # Final result with only EPIs
    else: # Else follow Petrick's method for further simplification
        P = [[findVariables(j) for j in chart[i]] for i in chart]
        while len(P)>1: # Keep multiplying until we get the POS form of P
            P[1] = multiply(P[0],P[1])
            P.pop(0)
        final_result = [min(P[0],key=len)] # Choosing the term with minimum variables from P
        final_result.extend(findVariables(i) for i in EPI) # Adding the EPIs to final solution
    print('\n\nSolution: Expression(min) = '+' + '.join(''.join(i) for i in final_result))

    input("\nPress enter to exit...")
```

same is upload on github.com on @Gagandeep-coep user name