VNUHCM-UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Cryptarithmetic Problem

Teacher: Nguyễn Ngọc Thảo

TA: Lê Ngọc Thành

Members:

21127430 – Nguyễn Huy Thành

21127162 – Lê Nguyên Thái

21127676 – Phan Hữu Phước

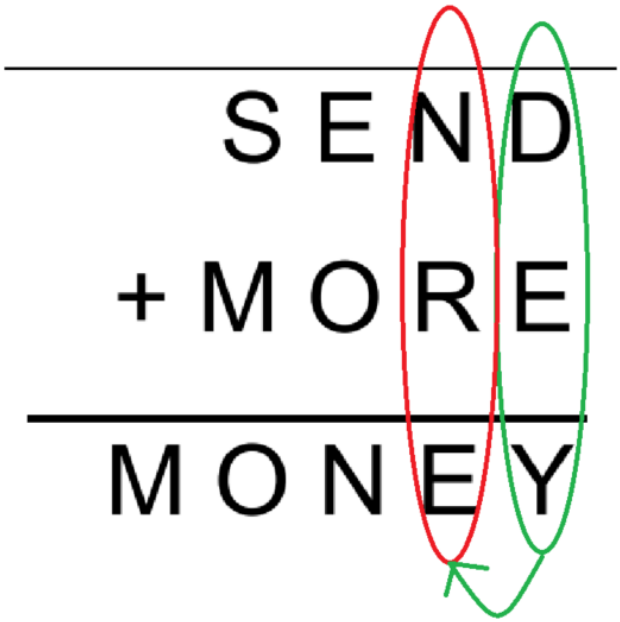
21127648 – Nguyễn Nhật Nam

Contents

1. Tổng quan

- Ở cả 4 level em đều dùng chung cho một thuật toán và cách tiếp cận của em đó chính là dạng bài CSP bằng cách chia bài toán thành constraint các cột đơn vị, chục, trăm, nghìn,... để giải quyết từng cột cho chúng sau đấy sẽ sử dụng kết quả đạt được ở cột thấp hơn để giải quyết ở cột cao hơn. Nếu tìm với cách gán hiện tại không thỏa thì sẽ backtracking ngược lại để tìm giá trị của biến phù hợp. Ví dụ: các biến được gán giá trị ở cột đơn vị sẽ vẫn được sử dụng cho cột chục, cột trăm,... và kèm theo đó là sự ảnh hưởng của carry ở các cột trước sẽ cộng dồn vào.

  Lý do chi tiết vì sao em lại dùng cách đi từ hàng đơn vị để bắt đầu đi lên sẽ được trình bày ở sau ạ!



Thực hiện và xử lý:

Tiền xử lý phép tính:

- **Bước 1**: Em sẽ nhập dữ liệu từ file ‘input.txt’ gồm nhiều chuỗi phép tính là và sẽ xử lý chúng. Sẽ gồm có 4 file input.txt, mỗi file sẽ có 5 test và tương ứng với từng level.
  - Thực hiện:

```
919  def main():
920
921      f = open('input.txt','r')
922      pheptinh = ''
923      cnt = 0
924      while True:
925          data = f.readline()
926          if data == '':
927              break
928          pheptinh = data.strip()
```

- Em sẽ dùng biến f để đọc dữ liệu từ file input.txt và em sẽ đọc theo từng dòng sau đó dữ liệu sẽ được đưa vào biến pheptinh để cho việc xử lý sau đó. Vòng lặp sẽ được dừng khi dữ liệu đọc ra từ file là một chuỗi rỗng.

- **Bước 2**: Em sẽ sử dụng một biến map toàn cục list_dif_zero={} để lưu những chữ cái ở đầu của mỗi hạng tử trước dấu bằng. Để khi thực hiện việc tìm kiếm thì em sẽ không cho những chữ cái này được phép mang số 1.

```
5   # Dùng để lưu những chữ cái ở đầu mỗi hạng tử
6   list_dif_zero ={}
```

- Gọi trong hàm main để xử lý:

```
930          solve_khac_khong(pheptinh)
```

- Thân hàm:

```
865  def solve_khac_khong(pheptinh):
866      global list_dif_zero
867      check = False
868      for u in pheptinh:
869
870          if(check==False):
871              list_dif_zero[u]=1
872              check=True
873          if(u<'A' or u>'Z'):
874              check=False
875              continue
876      return
```

- Với mỗi chữ cái gặp được đầu tiên sau những đợt bị ngắt quãng bởi các dấu +,-,*,(,) hay nói cách khác là những kí tự khác chữ cái thì chữ cái gặp được đầu tiên sẽ chính là chữ cái ở đầu của mỗi hạng tử.
- Tiếp đến em chỉ cần gán cặp {key,value} với key là chữ cái còn value = 1 (đánh dấu) ở dòng 871.

- **Bước 3**: em sẽ sử dụng một biến toàn cục map mp={} để có thể chuyển một kí tự tồn tại trong phép tính thành một số nguyên đại diện duy nhất. Và đồng thời em cũng sử dụng thêm một biến map toàn cục mp_nguoc={} để chuyển ngược từ số nguyên đại diện về lại thành kí tự. Để về sau có thể thuận tiện cho việc xử lý.
  - Khai báo:

```
8   # Biến map dùng để map chữ cái với một số nguyên đại diện
9   mp = {}
10
11  # Biến map dùng để map số nguyên đại diện về ngược lại chữ cái
12  mp_nguoc ={}
```

  - Thực hiện:

```
935         # map with interger
936         count = -1
937         for u in pheptinh:
938             if(u>='A' and u<='Z'):
939                 if u not in mp:
940                     count +=1
941                     mp[u]=count
942                     mp_nguoc[count]=u
```

  - Em dùng một biến count để làm số nguyên đại diện cho chữ cái. Nếu gặp một chữ cái mà chưa xuất hiện biến mp thì em sẽ tăng biến count lên và gán count làm số nguyên đại diện cho kí tự đó và đồng thời cũng gán ngược lại cho biến mp_nguoc={} để tạo song ánh giữa chữ cái và số nguyên đại diện cho nó.
- **Bước 4**: Em sẽ tách làm 2 phần chuỗi là phần chuỗi trước dấu bằng và sau dấu bằng của phép tính
  - Thực hiện tách chuỗi sau dấu '=':

```
945         # Tách kết quả sau dấu bằng
946         kq = ""
947         check = False
948         for u in pheptinh:
949             if(check == True and u>='A' and u<='Z'):
950                 kq = kq + u
951             if(u=='='):
952                 check = True
953
```

- Em sẽ chỉ lấy những kí tự sau khi đã duyệt thấy dấu bằng '='. Kết quả được lưu trong biến kq
- Thực hiện tách chuỗi trước dấu '=':

```
954         # Tách các hạng tử trước dấu bằng
955         new_pheptinh=""
956         for u in pheptinh:
957             if( u=='='):
958                 break
959             else:
960                 new_pheptinh = new_pheptinh + u
961         pheptinh = new_pheptinh
```

- Em sẽ lấy hết tất cả các kí tự cho đến khi gặp dấu bằng thì sẽ ngắt vòng. Kết quả sẽ được lưu vào biến new_pheptinh. Nhưng đó em gán lại cho biến pheptinh (dòng 961) để dễ cho việc gọi hàm ở phía sau.

- **Bước 5**: em sẽ xác định xem biểu thức này có chứa dấu ngoặc '(' ,')' hay có chứa dấu nhân '*' để có thể xử lý cho từng trường hợp một cách đúng đắn
  - Thực hiện:

```
963         # Xem xét level
964         check_dau_nhan=False
965         check_dau_ngoac= False
966         for u in pheptinh:
967             if(u=='*'):
968                 check_dau_nhan = True
969             if(u=='(' or u==')'):
970                 check_dau_ngoac = True
```

  - Em dùng 2 biến check cho dấu ngoặc và dấu nhân mỗi khi tìm thấy chúng trong phép tính.

- **Bước 6**: Tùy thuộc vào 2 biến check_dau_nhan và check_dau_ngoac sẽ dẫn đến một tiền xử lý khác nhau cho hai level 3 và level 4. Và ngay trong bước này, toàn bộ dấu ngoặc sẽ được phá bỏ hoàn toàn và việc nhân đa thức cho level 4 cũng sẽ được giải quyết một cách triệt để. Em sẽ nói rõ điều này trong hai phần của Level 3 và Level 4.
- **Bước 7**: Đây là bước tiền xử lý quan trọng để cấu trúc phép tính SAU KHI THỰC HIỆN BƯỚC 6 đối với level3 và level4, để tạo thành các constraint dạng cột. Em sẽ trình bày trước với phép cộng trừ '+ -' ở các level 1,2,3. Còn về phép nhân ở trong hàm nensolv4() thì sẽ nói ở phần Level 4.
    - Gọi hàm thực thi nensolv4():
    -
```
994         #Nén số + Tái cấu trúc lại các cột constraint
995         c = nensolv4(pheptinh,kq)
```
    - Thân hàm và cách thực hiện:
    -
```
675 def nensolv4(pheptinh,kq):
676     global list_dif_zero
677     global size
678 
679     list_dif_zero[kq[0]]=1
680 
681     #Tách phép tính thành các tuple hạng tử
682     #đi kèm với dấu của hạng tử
683     a = tachhangtu(pheptinh)
684 
685     #Đo lường size
686     size = sys.getsizeof(a)+sys.getsizeof(pheptinh)
```
    - Trong hàm, sẽ đánh dấu lại kí tự đầu tiên của của chuỗi kết quả để khi sau này backtracking kí tự này sẽ không được phép mang giá trị số 1.
    - Kế tiếp, em sẽ đi tách phép tính trước dấu '=' thành một loạt các tuple(hệ_số, hạng tử) và lưu lại thành danh sách trong biến a. Cách thực hiện của hàm này hoạt động như sau:

```python
def tachhangtu(a):
    b = []
    n= len(a)
    term = False
    hangtu=""
    for i in range(n):
        if(term==False):
            hangtu=hangtu+a[i]
            term=True
        else:
            if(a[i]=='+' or a[i]=='-'):
                if(a[i-1]=='*'):
                    hangtu=hangtu+a[i]
                else:
                    term=False
                    dau=1
                    new_hangtu=""
                    for u in hangtu:
                        if(u=='-'):
                            dau*=-1
                        else:
                            if(u!='+'):
                                new_hangtu= new_hangtu+u
                    b.append([dau,new_hangtu])

                    hangtu=a[i]
            else:
                hangtu= hangtu+a[i]

    dau=1
    new_hangtu=""
    for u in hangtu:
        if(u=='-'):
            dau*=-1
        else:
            if(u!='+'):
                new_hangtu= new_hangtu+u
    b.append([dau,new_hangtu])
    return b
```

- Trong hàm tách các hạng tử này, thì phép tính(dạng chuỗi) được truyền vào là biến a.
- Danh sách để lưu kết quả là list b
- Bên cạnh đó còn sử dụng thêm biến cờ term để kiểm tra xem ngay trước thời điểm hiện tại có đang duyệt trong một hạng tử nào hay không? Nếu bằng False là không, True là có

- Biến hangtu="" và dau=1 chính là chuỗi hạng tử và dấu của hạng tử đang xét.
- Ta sẽ duyệt qua chuỗi a:
  - Nếu như biến term=False nghĩa là ngay trước hiện tại đang không duyệt hạng tử nào => phần tử a[i] hiện tại chính là kí tự đầu tiên của hạng tử mới (vì các hạng tử nối tiếp nhau liên tục) và ta gán term = True để báo hiệu cho việc ta đang được duyệt một hạng tử.
  - Ngược lại nếu biến term = True thì ngay trước hiện tại ta đang duyệt trên một hạng tử. Tiếp theo, ta kiểm tra kí tự hiện tại có phải là dấu cộng hay dấu trừ? Nếu không, thì có nghĩa là vẫn ở trong một dãy kí tự liên tục, nếu có thì ta phải kiểm tra thử xem kí tự phía trước là dấu '*' hay không?
    - Nếu là dấu nhân '*', thì hạng tử này chính là một dãy các chữ số nhân nhau. Ví dụ: A*B*-C. thì ta chỉ cần thêm các kí tự này vào.
    - Nếu không phải là dấu nhân '*'=> đây chính là kết thúc của hạng tử vừa mới xét trước đó. Ví dụ: ABC+BC. Ta cần gán lại giá trị của term = False để báo hiệu vừa kết thúc một lượt duyệt hạng tử. Kế tiếp ta sẽ xử lý RÚT GỌN cho hạng tử vừa thu được trong biến hangtu:
      - Ta sẽ tính dấu của hạng tử này bằng cách với mỗi dấu trừ ta gặp trong hạng tử này thì sẽ nhân biến dau với -1
      - Và kèm theo đó một hạng tử mới new_hangtu được thành lập sẽ được sinh ra bằng cách loại bỏ các dấu cộng trừ trong hạng tử đó.
      - Ví dụ: -A*-B*-C thì kết quả của rút gọn là: new_hangtu=A*B*C và dau=-1
      - Kết quả sẽ được để vào list b
- Sau khi lặp hết vòng for thì ta phải giải quyết cho hạng tử cuối cùng của mảng vì ở hạng tử này trong vòng lặp for không có điều kiện để giải quyết hạng tử này.
- Hàm sẽ trả về list b
- Quay trở lại hàm nensolv4()

```
    b = [[] for _ in range(100)]
    # Cấu trúc lưu sẽ là:
    # - Symbol - hệ số - chữ cái(bình thường)/bảng tần suất(phép nhân)
    n = len(a)
    # gom nhom
    b_max = 0
    for u in a:
        hangtu = u[1]
        heso = u[0]
        check_nhan = False
        for v in hangtu:
            if(v=='*'):
                check_nhan=True
                break
        symbol = '+'
        if(check_nhan):
            symbol='*'
            zero = 0
            new_hangtu = []
            for v in hangtu:
                if(v=='0'):
                    zero +=1
                if(v>='A' and v<='Z'):
                    new_hangtu.append(v)
            new_hangtu.sort()
            # Nhóm từng nhân tử trong hạng tử
            dem = 1
            m = len(new_hangtu)
            tanso = []
            for j in range(1,m):
                if(new_hangtu[j] == new_hangtu[j-1]):
                    dem+=1
                else:
                    tanso.append([new_hangtu[j-1],dem])
                    dem=1
            tanso.append([new_hangtu[m-1],dem])
            b[zero].append([symbol,heso,tanso])

            # Tìm ra số lượng cột tối đa cần xem xét
            b_max = max(b_max,zero+1)
        else:
            #Hạng tử này chỉ là hạng tử bình thường (không có dấu nhân)
            symbol='+'
            m=len(hangtu)
            for j in range(m):
                letter = hangtu[j]
                tmp= []
                tmp.append([letter,0])
                cot = m - j - 1
                b[cot].append([symbol,heso,tmp])
                # Tìm ra số lượng cột tối đa cần xem xét
                b_max = max(b_max,cot+1)

```

- Tiếp theo trong hàm này, em sẽ tạo các constraint theo từng cột đơn vị, chục,... dựa trên danh sách a vừa thu được từ việc tách các hạng tử.
- Em dùng một list b 2 chiều để lưu từng mảng giá trị cho từng constraint.
- Trong vòng lặp em sẽ lấy ra từng hạng tử cũng như dấu đi kèm với hạng tử đó.
- Kế tiếp em sẽ kiểm tra xem hạng tử này có phải là hỗn hợp của các phép nhân hay không? Ví dụ: A*B*C. Vì ở đây em sẽ chỉ nói về phép cộng trừ (+,-) còn về phải nhân em sẽ để ở phần Level 4.
- Bắt đầu từ dòng 726. Sau khi cờ kiểm tra báo hiệu hạng tử này không là hỗn hợp của các phép nhân mà chỉ là hạng tử bình thường (ví dụ: ABC, DEF).
    - Em sẽ lặp qua từng chữ cái trong hạng tử này và tạo một mảng tmp sau đấy để vào trong đó một mảng nhỏ gồm 2 phần tử [chữ cái,0]. Mục đích em làm thế này là để phù hợp với format của bên hạng tử hỗn hợp có phép nhân.
    - Tiếp theo em sẽ đi tính cột constraint (đơn vị, chục, trăm ,...) mà KÍ TỰ của HẠNG TỬ này ảnh hưởng trực tiếp bằng cách tính:
        - ⇨ cot = m-j-1 trong đó m là chiều dài của hạng tử và j là vị trí của hạng tử.
        - ⇨ Ví dụ: hangtu = ABCD. Khi xét để kí tự B thì cột constraint mà kí tự B ảnh hưởng trực tiếp sẽ là 4 – 1 – 1 = 2. Mà ta có mảng b bắt đầu với cột 0, thì cột mà kí tự B ảnh hưởng trực tiếp là cột 2 hay nói cách khác là cột trăm (100). Tương tự cho kí tự A sẽ ảnh hưởng trực tiếp đến cột nghìn, C ảnh hưởng đến cột chục và D ảnh hưởng đến cột đơn vị.
    - Kế tiếp, em sẽ đẩy vào danh sách b ở constraint thứ cot vào b[cot] là một mảng con [symbol,heso,tmp]. Trong đó:
        - symbol là kí tự = '+' để phân biệt với kí tự symbol = '*' nếu là hạng tử hỗn hợp.
        - Heso là dấu của kí tự này hay nói các khác cũng chính là dấu của hạng tử đang xét.
        - Tmp là mảng nhỏ [letter,0] trong letter là kí tự đang xét, còn số nguyên 0 ở phía sau thì không có tác dụng gì cả chỉ để tạo thành dạng giống với phép nhân của hạng tử hỗn hợp sau này.
- Kế tiếp ngay trong hàm nensolv4():

```
742     n = max(b_max,len(kq))
743     c = [[] for _ in range(n)]
744 
745     for i in range(n):
746         b[i].sort(key = lambda x: x[2])
747         m = len(b[i])
748         tong = 0
749         if(len(b[i])!=0):
750             tong = b[i][0][1]
751         for j in range(m-1):
752             if(b[i][j][2]==b[i][j+1][2]):
753                 tong+=b[i][j+1][1]
754             else:
755                 if(tong != 0):
756                     c[i].append([b[i][j][0],tong,b[i][j][2]])
757                 tong = b[i][j+1][1]
758 
759         if(tong!=0): c[i].append([b[i][m-1][0],tong,b[i][m-1][2]])
760         #Them ket qua
761         n1 = len(kq)
762         if(i<n1):
763             c[i].append(['+',1,kq[n1-i-1]])
764         else:
765             c[i].append(['+',1,'0'])
766 
767     size += sys.getsizeof(c)
```

- Việc làm tiếp theo của em đó chính là tính tổng hệ số của các chữ cái giống nhau trong cùng 1 cột Constraint. Ví dụ cột trăm gồm: A+B-A+B=0*A+2*B=2*B. Với mục đích để có thể giảm được độ phức tạp của bài toán xuống.
- Em sẽ khởi tạo một mảng c hai 2 chiều tương tự mảng b với kích thước sẽ là số cột constraint tối đa có thể có.
- Tiếp đến, em sẽ lặp qua n cột constraint của mảng b (đã tính phía trên). Với mỗi mảng b[i] tương ứng với danh sách các chữ cái và hệ số kèm theo của cột constraint thứ i. Thì em sẽ sort lại mảng này theo cặp [kí tự, 0], vì tham số x[2] trong hàm sort (dòng 746) là một mảng nhỏ gồm 2 phần tử. Kết quả sau khi sort sẽ ưu tiên theo chữ cái ví dụ: (A,0),(A,0),(B,0),(C,0).
- Kế tiếp, em sẽ tính tổng tong cho các hệ số của các kí tự bằng cách cộng dồn hệ số của chúng nếu như kí tự phía trước giống với kí tự phía sau. Nếu kí tự trước và kí tự sau khác nhau tự ta đã kết thúc việc dồn hệ số cho một kí tự nên sẽ thêm vào mảng c tương tự như cách thêm ở mảng b (đã giải thích phía trên) với cùng 1 format.
- Sau đó, em thêm và cuối danh sách của từng cột constraint là một mảng con của kí tự của chuỗi kết quả với cột tương ứng.
- Và nếu như ở những phần mà độ dài của các hạng tử không đồng đều thì em sẽ thêm kí tự ‘0’ đại diện vào cấu trúc để tiện cho việc xử lý.

⇨ Vậy là ta đã hoàn thành xong việc tiền xử lý cho phép cộng và phép trừ bằng một mảng cấu trúc 2 chiều với mỗi chiều ứng với một constraint sẽ có các mảng nhỏ chứa [symbol,hệ số, [kí tự , 0]] với các kí tự ở phần kết quả thì ta có thể để ở dạng [symbol,hệ số, kí tự] vì có thể phân biệt với các trường hợp còn lại.

2. Level 1, 2

- Level 1:
  Theo quan sát ban đầu thì em nhận thấy một cột đơn vị, chục, trăm,.... đều sẽ bị ảnh hưởng bởi 5 yếu tố đó là:
  - 2 biến của 2 hạng tử cộng ở phía trên gọi là A,B
  - 1 biến của kết quả ở phía dưới gọi là R
  - 1 biến carry từ các cột đằng trước cộng dồn lên, gọi là x1
  - 1 biến carry do kết quả hiện tại tạo ra ảnh hưởng, gọi là x2

  Nhưng nếu để ý kỹ thì ta chỉ cần quan tâm 3 biến đó là:
  - 2 biến của 2 hạng tử cộng ở phía trên
  - 1 biến carry x1 từ các cột đằng trước cộng dồn lên

  Vì ta có biến R và x2 có thể được suy ra một cách duy nhất từ 3 biến A,B,x1

  $R = (A+B+x1)\%10$ và $x2= (A+B+x1)//10$.

  Trong đó $x1,x2 =\{-1,0,1\}$

- Level 2:

  Về mặt tổng quát hơn so với level 1, thì khi lúc này sẽ có nhiều biến ở hạng tử phía sẽ cùng tham gia với nhau. Nhưng các tính chất so với Level 1 vẫn sẽ giữ nguyên là ta chỉ cần xác định các biến cùng tham gia của mỗi hạng tử (A,B,C,D,...) và carray x1.

  $R = (A+B+C+D+....+ x1)\%10$, $x2 = (A+B+C+D+....+ x1)\%10$

Trong đó carry x1 có thể mang giá trị rất lớn. Và đây cũng chính là lí do tại sao em lại chọn theo phương pháp ép constraint từ cột đơn vị (phải nhất) sang cột to nhất (trái nhất) chứ không phải là bắt đầu ép constraint từ cột to nhất (trái nhất) sang cột đơn vị (phải nhất).

- Vì nếu như ta ép constraint từ cột trái nhất thì lúc xét ở cột này ta sẽ bị thiếu dữ kiện của biến carry x1 (vì ta chưa xác định được).
- Dẫn đến việc ta phải lặp và tìm trong các trường hợp có thể có của carry x1.

- Nhưng với bài toán mà có quá nhiều tham số biến ở phần hạng tử (A,B,C,...) thì lúc này khoảng giá trị carry x1 có thể mang sẽ rất lớn.
  - Ví dụ: trước dấu bằng có tới 1000 hạng tử thì giá trị carry x1 có thể mang là khoảng [-900;900].
- Nếu như ta cứ phải thay thế lần lượt các giá trị của carry x1 như thế này thì độ phức tạp thuật toán sẽ rất lớn.

Vậy nên việc khi ta ép constraint từ cột phải nhất (cột đơn vị) sang cột trái nhất thì lúc này giá trị carry x1 sẽ bằng 0 và ta có thể dễ dàng suy ra một cách chính xác carry x2 cho constraint của cột tiếp theo mà không cần phải quan tâm việc tìm carry ở cột này.

Thực thi:

- Gọi hàm solve(c,cnt):

```
983             solve(c, cnt)
```

- Thân hàm:

```python
def solve(all_constraint, cnt):
    ans = [-1 for _ in range(11)]
    diff = [0 for _ in range(10)]
    f = open('output.txt', 'a')
    #Đo lường thời gian bắt đầu 1 test
    start_time = time.time()
    global size
    global ketqua
    global mp
    global list_dif_zero
    global mp_nguoc
    if(chonbien(all_constraint,0,0,ans,diff) == True):
        ans = []
        for i in range(len(ketqua)):
            if(ketqua[i]==-1):
                continue
            else:
                letter = mp_nguoc[i]
                ans.append((letter,ketqua[i]))

        #Đo lường thời gian kết thúc 1 test
        end_time = time.time()
        ans.sort()
        for i in range(len(mp)):
            f.write(str(ans[i][1]))
        f.write("\n")
        print("Time ",cnt,": ",end_time - start_time," s")
    else:
        f.write("NO SOLUTION\n")

    # Delete previous data
    size=0
    mp.clear()
    mp_nguoc.clear()
    list_dif_zero.clear()
    ketqua.clear()
    f.close()
    return
```

- Ở hàm này tham số truyền vào là toàn bộ constraint các cột mà ta đã làm ở tiền xử lý. Và mục tiêu chỉ đơn giản là khởi tạo các biến để chuẩn bị cho việc thực hiện hàm chonbien hay nói các khác là backtracking để tìm

biến. Trong đó hàm chonbien sẽ trả về True nếu tìm thấy kết quả, False nếu không tìm thấy kết quả.

- Trong hàm chonbien()

```
644  def chonbien(c,constraint,carry,ans,diff):
645      global ketqua
646      # Kiểm tra xem đã thỏa mãn hết constraint chưa?
647      if(constraint==len(c)):
648
649          # Kiểm tra xem carry cộng dồn bắt buộc phải = 0
650          if(carry==0):
651              ketqua=ans
652
653              return True
654          else:
655              return False
656      list= c[constraint]
657
658      #recursive
659      i=0
660      sum_left=carry # Se mang carray cua truoc
661      return forward_checking(i,0,list,sum_left,ans,diff,constraint,c,1,0)
662
```

  - Trong hàm này tham số truyền vào sẽ là danh sách toàn bộ constraint biến c, thứ tự của cột constraint là biến constraint , biến carry để truyền biến nhớ từ cột constraint phía trước, biến ans để lưu giá trị được gán vào trong số nguyên đại diện của kí tự đó.
    - Ví dụ: chữ A có số nguyên đại diện là 4 , và A được gán giá trị 8. Thì ans[4] = 8. Còn nếu chữ B có số nguyên đại diện là 2 là chưa được giá trị thì ans[2]=-1

    Và biến diff để đánh dấu các biến chưa bị gán:

    - Ví dụ: nếu giá trị 4 đã được gán giá trị rồi thì diff[4]=1. Ngược lại nếu giá trị 5 chưa được gán trị cho biến nào hết thì diff[5]=0.
  - Trong hàm trên ta có trường hợp suy biến sẽ là, biến constraint = len(c). Có nghĩa là kiểm tra xem thứ tự của cột constraint hiện tại đã vượt quá số lượng của cột constraint thực tế hay chưa? Vậy nếu biến constraint = len(c) có thì có nghĩa là nó đã thỏa hết tất cả các constraint và ta chỉ cần kiểm tra biến carray do constraint đằng trước mang đến phải bằng 0 thì ta sẽ có kết quả đúng, còn nếu không thì đây là một phép gán sai.
  - Biến list = c[constraint] sẽ là một mảng các [symbol,hệ số,[kí tự,0]] của cột constraint hiện tại đang xét.

    - Tiếp đến ta sẽ có tổng của cột constraint đang xét thì ngay từ lúc bắt đầu xét constraint thì nó đã chịu ảnh hưởng bởi carry của cột constraint phía trước.
    - Sau đấy ta sẽ gọi hàm forward_checking để tìm kiếm cũng như loại bỏ các giá trị trên đường đi backtracking của nó.
- Lưu ý: mối quan hệ giữa hàm forward_checking() và hàm chonbien() là hỗn tương
- Hàm forward_checking: Vì đây là level 2 nên em không giải thích phần của phép nhân trong hàm forward_checking này.

```python
def forward_checking(i,x,list,sum,ans,diff,order_constraint,c,value_nhan,heso_tmp):
    global mp
    global list_dif_zero
    symbol='+'
    if(i<len(list)):
        symbol = list[i][0]
    if(symbol=='+'):

        if(i!=len(list)-1):
            letter = str(list[i][2][0][0])
        else:

            letter = str(list[i][2])
        heso = list[i][1]
        if(letter=='0'):
            giatri=0
        else:
            order = mp[letter]
            giatri = ans[order]
        du=sum%10
        if(i==len(list)-1):
            #Cuối constraint
            #Kiểm tra xem chữ cái cuối này được gán hay chưa
            if(giatri==-1):
                # Nếu chữ cái kết quả chưa được gán thì sẽ gán
                #Để thỏa constraint này thì cần phải giá trị dư cần phải chưa bị chi
                if(diff[du]==0):
                    if(du==0):
                        if letter in list_dif_zero:
                            return False
                    # Nếu thỏa thì set up cho constraint kế tiếp
                    diff[du]=1
                    ans[order]=du
                    carry = sum//10
                    check = chonbien(c,order_constraint+1,carry,ans,diff)
                    if(check==True):
                        return True
                    ans[order]=-1
                    diff[du]=0
                else:
                    # Với cách gán hiện tại thì sẽ không thỏa mãn constraint này vì
                    return False
```

- Hàm sẽ gồm các tham số như sau:
  - Biến i: thứ tự của chữ cái trong danh sách của cột constraint hiện tại.
    - ⇨ Ví dụ: i=2, thì list[i] = [+,3,[‘A’,0]]
  - Biến x: Sẽ giải thích trong phép nhân
  - Biến list: chính là danh sách của cột constraint đang xét

- Biến sum: Tổng giá trị của cột constraint hiện tại.
  - ⇨ Ví dụ: Đang xét ở cột constraint thứ 2 hàng trăm thì ta có tại constraint này A+ 2*B sau khi gán có được 1+2*2=5 => sum=5. Nhưng thực tế 5 hàng trăm thì sẽ là 500.
- Biến ans: Lưu lại kết quả của các biến (đã nói ở hàm chonbien).
- Biến diff: Đánh dấu giá trị được gán (đã nói ở hàm chonbien).
- Biến order_constraint: thứ tự của cột constraint hiện tại đang xét
- Biến c: Lưu lại toàn bộ constraint
- Biến value_nhan và heso_tmp: sẽ giải thích ở phần Level 4

- Nếu gặp symbol là dấu cộng '+' thì ta sẽ giải quyết theo hướng của các kí tự hạng tử bình thường.
- Việc em check từ dòng 528 -> 532 là do vấn đề cuối danh sách thì kí tự kết quả sẽ có format lưu mảng khác (em đã có nói ở phần tiền xử lý ạ).
- Em sẽ lấy chữ cái và hệ số của từ danh sách.
- Em sẽ kiểm tra chữ cái hiện tại là chữ số 0 thì bằng 0 ngược lại thì em sẽ dùng biến mp để lấy ra số nguyên đại diện => lấy trị từ mảng ans.
- Tiếp đến em sẽ kiểm tra là cuối danh sách của cột constraint hiện tại hay chưa?
  - Nếu rồi thì sẽ kiểm tra chữ cái hiện tại đã được gán trị hay chưa?
    - ⇨ Nếu rồi thì sẽ kiểm tra xem chữ số hàng đơn vị của sum hay chính là du= sum%10 có đã được chọn hay chưa?
      - Nếu được chọn rồi thì phải kiểm tra chữ cái này có phải là chữ cái xuất ở đầu của một cụm kí tự nào đó hay không. Nếu thỏa thì sẽ gán giá trị cho chữ cái này là tăng lên constraint tiếp theo bằng cách gọi hàm chonbien()
      - Nếu chưa được chọn thì trả về False vì không còn cách nào thỏa

```
563                     ans[order]=-1
564                     diff[du]=0
565                 else:
566                     # Với cách gán hiện tại thì sẽ không thỏa mãn constraint này vì có giá trị cho chữ
567                     return False
568             else:
569                 #Nếu đã được gán giá trị trước đó rồi thì kiểm tra kết quả với sum hiện tại
570                 if(du==giatri):
571                     #Thoả điều kiện với cách gán này
572                     carry=sum//10
573                     return chonbien(c,order_constraint+1,carry,ans,diff)
574                     #Vì giá trị đã được gán từ một vòng lặp trước đó nên ta không cần gán reverse lại
575                 else:
576                     # Không thỏa điều kiện
577                     return False
578 
579         else:
580             if(giatri!=-1):
581                 # Vì biến này đã được gán từ trước nên ta không thể thay đổi giá trị của biến này được
582                 check=forward_checking(i+1,0,list,sum+giatri*heso,ans,diff,order_constraint,c,1,0)
583                 if(check == True):
584                     return True
585             else:
586                 start = 0
587                 if letter in list_dif_zero:
588                     start = 1
589                 for j in range(start,10):
590                     if(diff[j]==0):
591                         ans[order] = j
592                         sum = sum + j*heso
593                         diff[j]=1 # occupied
594                         check = forward_checking(i+1,0,list,sum,ans,diff,order_constraint,c,1,0)
595                         if(check == True):
596                             return True
597                         ans[order] = -1
598                         sum = sum - j*heso
599                         diff[j]=0 # unoccupied
600     return False
```

- Nếu giá trị chữ cái hiện tại đã được gán và nếu bằng với chữ cái hàng đơn vị thì sẽ đi tiếp. Nếu không thì trả về False.
- Nếu hiện tại chúng ta đang xét cho kí tự chưa phải cuối cùng (ở dòng 580) thi có nghĩa là các chữ cái của hạng tử tham gia vào việc cộng trừ. Thì ta vẫn sẽ xét giống như các bước phía trên.
- Nếu giá trị của biến này đã được gán rồi thì sẽ gọi đệ quy cho kí tự của hạng tử kế tiếp xử lý. và kèm theo đó phải cộng sum cho giá trị được gán với hệ số tương ứng.
- Còn nếu như giá trị chưa được gán thì cần kiểm tra xem biến chữ cái này có cần khác 1 (vì là đầu chuỗi) hay không?Sau đấy sẽ lặp qua các giá trị chưa được gán trong mảng diff.
    - Nếu diff[i] =0 thì giá trị i chưa được gán.
    - Ta cần gán ans[order] là mảng chứa kết quả với order là số nguyên đại diện cho chữ cái hiện tại (đã giải thích phía trên).

- Cộng giá trị sum cho một lượng là j*heso là tổng của chữ cái hiện tại tác động lên cột constraint.
- Sau đấy chỉ cần gọi forward_checking() cho hạng tử tiếp theo (i+1)

- Và lưu ý sau mỗi lần gọi đệ quy em sẽ khôi phục lại trạng thái trước đó để có thể đảm bảo tính đúng đắn cho các lượt search tiếp theo.

=> Khi tìm được kết quả thì bên hàm chonbien() sẽ trả về giá trị True và kèm theo mảng ketqua toàn cục được trả về hàm cho hàm solve()

```python
def solve(all_constraint, cnt):
    ans = [-1 for _ in range(11)]
    diff = [0 for _ in range(10)]
    f = open('output.txt', 'a')
    #Đo lường thời gian bắt đầu 1 test
    start_time = time.time()
    global size
    global ketqua
    global mp
    global list_dif_zero
    global mp_nguoc
    if(chonbien(all_constraint,0,0,ans,diff) == True):
        ans = []
        for i in range(len(ketqua)):
            if(ketqua[i]==-1):
                continue
            else:
                letter = mp_nguoc[i]
                ans.append((letter,ketqua[i]))

        #Đo lường thời gian kết thúc 1 test
        end_time = time.time()
        ans.sort()
        for i in range(len(mp)):
            f.write(str(ans[i][1]))
        f.write("\n")
        print("Time ",cnt,": ",end_time - start_time," s")
    else:
        f.write("NO SOLUTION\n")

    # Delete previous data
    size=0
    mp.clear()
    mp_nguoc.clear()
    list_dif_zero.clear()
    ketqua.clear()
    f.close()
    return
```

và sẽ in kết quả ra file output theo đúng thứ tự bảng chữ cái.

3. Level 3

Ở Level 3 này, ta sẽ quay trở lại phần tiền xử lý ở **bước 6** đã có đề cập ở trên để nói về việc xử lý ngoặc cho phép tính.
Bài toán sẽ được giải quyết nếu ta đưa được **chuỗi input có ngoặc** trở thành **chuỗi không có ngoặc**, lúc này bài toán sẽ đưa về level 2 và ta có thể dễ dàng giải quyết.
Ta sẽ sử dụng hàm **khungoac(str)** để đưa bài toán về level 2.
Hàm sẽ trả về chuỗi đã được xử lí khử ngoặc từ chuỗi str đầu vào.

Ta nhận thấy nếu trước dấu ngoặc là dấu trừ, thì khi phá sẽ **thay đổi toàn bộ dấu ở trong ngoặc**, còn dấu cộng thì **giữ nguyên**, vì vậy ta sẽ lưu một biến **reverse** để kiểm tra xem tại dấu đang xét có cần đổi hay không
Hàm sẽ sử dụng một biến index để duyệt qua mảng str như một **con trỏ**, tại mỗi vị trí, sẽ có **4 trường hợp** xảy ra:
Trương hợp 1:Vị trí đang xét là một chữ số:

Lúc này hàm sẽ truyền chữ số này vào mảng kết quả, sau đó tăng biến con trỏ lên một và duyệt trường hợp tiếp theo

```
if Alphabet(input[index]):
    res+=input[index]
    index+=1
    continue
```

Trường hợp 1

Trường hợp 2: Vị trí đang xét là dấu mở ngoặc "(":

Khi gặp dấu mở ngoặc, ta sẽ lưu biến **reverse** đang sử dụng vào **stack** và thay đổi biến **reverse** tùy theo **dấu ở trước dấu ngoặc**, sẽ **lật bit** lại nếu là **dấu trừ** và **giữ nguyên** nếu là **dấu cộng**.

```
if input[index] == '(':
    stack.append(reverse)
    if input[index-1] == '-':
        reverse = 1 - reverse
```

Trường hợp 2.1

Có hai trường hợp đặc biệt của trường hợp 2 này mà ta sẽ nói đến sau.

Trường hợp 3:Vị trí đang xét là dấu đóng ngoặc ")":

Lúc này ta sẽ thay đổi biến reverse trở lại trạng thái trước đó đã lưu vào trong stack.

```
if input[index] == ')':
    reverse = stack.pop()
```

Trường hợp 3

Trường hợp 4:Vị trí đang xét là dấu "+" hoặc "-":

Tùy vào biến **reverse** mà dấu sẽ bị đổi hoặc không, dấu sau khi đổi này được lưu vào biến tạm **temp**

Trường hợp 4.1: đây là dấu nằm trước một dấu mở ngoặc "(":

Lúc này thay vì gán giá trị của dấu vào **mảng kết quả** thì hàm sẽ duyệt tiếp tục, để công việc gán này cho **trường hợp của dấu mở ngoặc**.

Trường hợp 4.2: các trường hợp còn lại:

Ta sẽ gán giá trị được lưu ở biến tạm này vào **mảng kết quả** và xử lí như thường

```
if input[index] =='+' or input[index] == '-':
    temp = reversed(input[index]) if reverse else input[index]
    if input[index+1] =='(':
        #truong hop 4.1
        index+=1
        continue
    #truong hop 4.2
    res+=temp
    index+=1
    continue
```

Trường hợp 4

Như đã đề cập ở trên, nếu ta vội vàng gán giá trị của dấu vào **mảng kết quả** thì sẽ có trường hợp gán sai dấu vì tồn tại trường hợp đặc biệt : **-(-A))**, trong trường hợp này, nếu ta vội vàng gán dấu ở trước dấu ngoặc (dấu trừ) thì sẽ ra trường hợp sai dấu vì sau khi xử lí kết quả sẽ phải là **+A**, **không có dấu trừ.**

Ở trường hợp này ta sẽ chia ra làm 2 trường hợp nhỏ hơn đó là (A) và (-A):

Trường hợp 2.2: (A), hay nói cách khác là **trường hợp mà chữ số nằm ngay sau dấu ngoặc.**

Lúc này, hàm sẽ thêm dấu đã được lưu từ trước vào mảng kết quả và tiếp tục xử lí bình thường.

Trường hợp 2.3: (-A) hay nói cách khác là **trường hợp mà có dấu trừ nằm sau dấu ngoặc**, ở trường hợp này, khi phá ngoặc thì **dấu của chữ số** đầu

tiên sẽ bị **ngược với dấu ở ngoài dấu ngoặc**, Ví dụ : -(-A)) sẽ thành +A và +(-A) sẽ thành -A nên ta sẽ add dấu ngược với dấu đã lưu từ trước vào mảng kết quả, sau đó tăng biến đếm lên 2(bỏ qua xử lí dấu "-") và tiếp tục xử lí.

```
if Alphabet(input[index+1]):
    res+=temp
else:
    res+=reversed(temp)
    index+=2
    continue
```

Trường hợp 2.2 và 2.3

- Sau khi giải quyết xong bước tiền xử lý cho level 3 thì bước tiếp theo để giải bài toán của sẽ tương tự như Level 2.

4. Level 4

- Ý tưởng của phần này là em vẫn sẽ tiếp tục theo phương pháp CSP như cả 3 level trên. Phương pháp của em là sẽ nhân toàn bộ các nhân tử để có thể phá hết tất cả các dấu ngoặc.
- Mặc dù em có thể làm cách là không cần xét constraint từng cột mà chỉ xét mỗi điều kiện là các kí tự phải có giá trị khác nhau và khi đệ quy đủ 10 giá trị thì sẽ kiểm tra xem biểu thức đưa vào có đúng hay không? Nhưng em cảm thấy như vậy thì chẳng khác nào Brute Force cả và nó sẽ khác hẳn tinh thần của CSP trong một vấn đề cụ thể như thế này.
- Nên khi em code theo hướng là nhân toàn bộ nhân tử ra thì mới dẫn đến bài code của em có đến khoảng 1000 dòng do Level 4 là chủ yếu ạ.
- Và em biết là code theo hướng này thì độ phức tạp một phần nào đó ra lớn hơn nhiều so với cách Brute Force em đã đề cập phía trên do việc các nhân tử khi nhân lại với nhau sẽ dẫn đến việc bùng nổ theo cấp số mũ nhưng em vẫn quyết định theo hướng thuần của CSP.

- Triển khai ý tưởng như sau:

Ví dụ ta có: AB*CD = (A0+B)*(C0+D) = A0*C0 + A0*D + B*C0 + B*D

- Và đây chính là cách em sẽ phá dấu ngoặc cho các dấu nhân.
- Vì code của em có tới 1000 dòng nên việc giải thích kỹ từng đoạn code sẽ khiến cho bài report trở quá dài nên ở phần này em sẽ tóm tắt quá trình như sau:

- Tiếp nối phần tiền xử lý của phép nhân Level 4 ở **bước 6**:

```
972         # Nếu biểu thức có dấu nhân => Level 4
973         if(check_dau_nhan==True):
974             # Khử ngoặc và dấu trừ
975             pheptinh = eli_minus(pheptinh)
976             pheptinh = nhanphanphoi2sohang(pheptinh)
977             pheptinh = khu_dau_tru(pheptinh)
978             # #Chuyển về dạng đơn thức nếu như dang có phép nhân
979             pheptinh = dangdonthuc(pheptinh)
980             pheptinh = khu_dau_tru(pheptinh)
```

- Hàm eli_minus() giúp cho việc chỉnh lại dạng cho các kí tự và sẽ gọi đến hàm khu_dau_tru() để xóa những dấu trừ và ngoặc mà không bị cố định bởi dấu nhân. Đây là hàm có chức năng gần giống với hàm khungoac().

- Tiếp đến để có thể khai triển nhân tử thì ta cần phải nhân các nhân tử ở dạng đơn. Ví dụ như: ABC*CDE cần được ưu tiên trước so với ABC*(CE+EF). Để làm việc này thì ta sẽ cần đến hàm nhanphanphoi2sohang(). Hàm này về ý tưởng chung sẽ lặp qua các hạng tử có nhân tử dạng đơn như đã nói ở trên. Sau đấy sẽ sử dụng hàm nhan2so() để có thể nhân 2 nhân tử đơn lại với nhau.
    - Trong hàm nhan2so() , ta có kha khá vấn đề cần phải giải quyết nào là nhân (dạng đơn với dạng đơn) (ví dụ AB*CD) và (dạng đơn với dạng triển khai) ví dụ: (AB*(C00*D00)). chúng ta cần phải giải quyết những vấn đề đó sao cho chúng phải về dạng chỉ có một kí tự trong một số.
        - Ví dụ: A00*B00 hợp lệ. Nhưng AB0*C00 thì không hợp lệ.
- Quay trở lại nhanphanphoi2sohang() cũng nãy sinh rất nhiều vấn đề. Nào là nảy sinh về sau khi chuỗi mới được tạo ra thì sẽ lắp đầy khoảng nào của chuỗi cũ vì sự ảnh hưởng của các dấu cộng, trừ và dấu * ảnh hưởng rất nhiều và khả năng bị bug chỗ này rất cao.
- Kế tiếp ta sẽ gọi hàm khu_dau_tru() một lần nữa để đảm bảo rằng các dấu ngoặc không bị cố định bởi dấu trừ đã bị xóa. Ví dụ AB*(CD+DE) thì hàm khu_dau_tru sẽ không xóa được ngoặc mà phải
- Tiếp theo là hàm Chủ chốt: dangdonthu() hàm này sẽ biến biểu thức về đơn thức hoàn toàn không có dấu ngoặc. Các thực hiện của hàm đó chính là sẽ gọi hỗn tương với hàm nhan2dathuc() . Ví dụ ta có:
AB+(CD+AB)*(A+B*(B+C)) + CD - E thì lúc này hàm dangdonthu() sẽ gọi hàm nhan2dathuc() cho cặp biểu thức (CD+AB) và (A+B*(B+C)). Rồi tiếp tục ở đây thì nhan2dathuc() chưa xử lý được biểu thức (A+B*(B+C)) nên sẽ gọi

ngược lại hơn hàm dangdonthu() cho biểu thức A+B*(B+C). Lúc này hàm dangdonthu() chỉ có thể xử lý được A nhưng còn B*(B+C) thì không xử lý được nên sẽ tiếp tục gọi hàm nhan2dathuc() để với 2 cặp biểu thức cần giải quyết là B và (B+C). Ở lúc này vì cả 2 là dạng thức nên hàm nhan2dathuc() có thể xử lý được và chỉ cần gọi hàm nhan2bieuthuc() để nhân lần lượt từng nhân tử với nhau nhờ vào hàm nhan2so() đã phát biểu ở trên.

- Quá trình đệ quy qua lại đến khi gặp trường hợp suy biến của cả 2 hàm. với hàm dangdonthu() thì đương nhiên là trường hợp suy biến là không có dấu ngoặc. Còn trường hợp suy biến của nhan2dathuc() 2 cặp đa thức đầu vào phải đều là một biểu thức không chứa ngoặc.
- Tiếp đến là quá trình cấu trúc cho phép nhân.

```
699         if(check_nhan):
700             symbol='*'
701             zero = 0
702             new_hangtu = []
703             for v in hangtu:
704                 if(v=='0'):
705                     zero +=1
706                 if(v>='A' and v<='Z'):
707                     new_hangtu.append(v)
708             new_hangtu.sort()
709             # Nhóm từng nhân tử trong hạng tử
710             dem = 1
711             m = len(new_hangtu)
712             tanso = []
713             for j in range(1,m):
714                 if(new_hangtu[j] == new_hangtu[j-1]):
715                     dem+=1
716                 else:
717                     tanso.append([new_hangtu[j-1],dem])
718                     dem=1
719             tanso.append([new_hangtu[m-1],dem])
720             b[zero].append([symbol,heso,tanso])
721 
722             # Tìm ra số lượng cột tối đa cần xem xét
723             b_max = max(b_max,zero+1)
```

- Tiếp tục với hàm nensolv4() ở trên level 2 với phép nhân thì trước hết mỗi hạng tử sau khi nhân sẽ luôn có dạng hợp lệ (đã được nói ở trên).

- Vậy khi hạng tử ở dạng này thì t có thể dễ dàng xác định được cột constraint mà nó sẽ ảnh hưởng đó chính là số lượng kí tự '0' trong chuỗi của hạng tử đó. Ví dụ: A000*B00 có 5 số 0 trong hạng tử thì cột constraint mà nó sẽ ảnh hưởng trực tiếp là cột thứ 5.
- Đồng thời em cũng sẽ tổ chức như sau: là không quan tâm đến số 0 mà chỉ quan tâm đến các kí tự còn lại. Ta chỉ cần đếm số lượng kí tự giống nhau thì đó chính lại bậc với kí tự đó tại chính hạng tử này.
- Ví dụ: A000*A00*B00 thì ta sẽ có A với bậc 2 hay hiểu là A^2, B với bậc 1 hay hiểu là B^1.
- Sau đó sẽ sử dụng một mảng con tương tự ở phía trên để lưu trữ với dạng [symbol,heso,[[A,2],[B,1]]] với phần tử thứ 3 trong mảng con này sẽ là một danh sách các bậc tương ứng của các kí tự trong hạng tử này.
- Bước tiếp theo em sẽ cộng dồn hệ số giống như của 3 level trước.
- Bước giải quyết vấn đề:
- Tương tự như trong 3 level trước thì vẫn sẽ gọi hàm chonbien():

```
641   # Moi constraint se la mot hang chuc, don vi, tram
642   def chonbien(c,constraint,carry,ans,diff):
643       global ketqua
644       # Kiểm tra xem đã thỏa mãn hết constraint chưa?
645       if(constraint==len(c)):
646
647           # Kiểm tra xem carry cộng dồn bắt buộc phải = 0
648           if(carry==0):
649               ketqua=ans
650
651               return True
652           else:
653               return False
654       list= c[constraint]
655
656       #recursive
657       i=0
658       sum_left=carry # Se mang carray cua truoc
659       return forward_checking(i,0,list,sum_left,ans,diff,constraint,c,1,0)
```

- Trong hàm này hầu hết các tham số sẽ không có gì thay đổi so với các level khác nhưng. Ở chỗ return fordward_checking() có 3 tham số hằng lần lượt là 0,1,0 thì để giải thích rõ ta sẽ qua bên hàm fordward_checking() để giải thích các phần còn lại chưa giải thích hết.

```
518   def forward_checking(i,x,list,sum,ans,diff,order_constraint,c,value_nhan,heso_tmp):
```

  - Biến x: ở đây sẽ là số thứ tự của kí tự trong mảng con thứ i của cột constraint thứ order_constraint.
    - Ví dụ c[order_constraint] là một dãy các danh sách của các cột constraint này. tiếp đó c[order_constraint][i] sẽ chứa

một mảng con có 3 phần tử (đã giải thích ở trên) trong đó phần tử thứ 3 c[order_constraint][i][2] sẽ chứa danh sách các kí tự và bậc tương ứng của chúng.
c[order_constraint][i][2][x] sẽ là cặp giá trị gồm chữ cái và bậc của chữ cái đó trong hạng tử này.

- Biến value_nhan: sẽ tích của các giá trị của chữ cái ^ bậc trong hạng tử này.
- Biến heso_tmp: sẽ chứa hệ số của hạng tử đang xét
- **Thực thi:**

```
599     else:
600         #Không cần xử lý trường hợp cuối constraint hiện tại vì ở cuối danh sách là 1 giá trị đơn (khác nhân)
601         #Chọn xong giá trị cho các biến trong constraint nhân
602         letter = list[i][2]
603         heso = list[i][1]
604 
605         if(x==len(letter)):
606             check=forward_checking(i+1,0,list,sum+value_nhan*heso_tmp,ans,diff,order_constraint,c,1,0)
607             if(check==True):
608                 return True
609         else:
610             chucai = letter[x][0]
611             bac = letter[x][1]
612 
613             order = mp[chucai]
614 
615             giatri = ans[order]
616             if(giatri!=-1):
617                 #Đã được gán trước đó
618                 check=forward_checking(i,x+1,list,sum,ans,diff,order_constraint,c,value_nhan*(giatri**bac),heso)
619                 if(check==True):
620                     return True
621             else:
622                 #Chưa được gán
623                 start = 0
624                 if chucai in list_dif_zero:
625                     start = 1
626                 for j in range(start,10):
627                     if(diff[j]==0):
628                         ans[order]=j
629                         diff[j]=1
630                         check= forward_checking(i,x+1,list,sum,ans,diff,order_constraint,c,value_nhan*(j**bac),heso)
631                         if(check == True):
632                             return True
633                         diff[j]=0# Unoccupied
634                         ans[order]=-1
635     return False
```

- Ở đây ta không cần phải xử lý trường hợp suy biến của cột constraint khi ta đang gặp số có dấu nhân vì nhân kết quả ở cuối đầu sẽ thuộc về dạng bình thường (đã xử lý trên symbol=’+’ rồi).
- Em sẽ cắt nhỏ từng mảng con ra như đã giải thích ở trên để dễ cho việc thao tác. Trong trường hợp này letter = c[order_constraint][i][2].
- Em sẽ kiểm tra xem với giá trị của x = len(letter) là có nghĩa ta đã xử lý xong trường hợp hạng tử này và ta sẽ có cập nhật lên cho hạng tử tiếp theo (i+1) và ở lúc này thì giá trị nhân đến hiện tại * hệ số sẽ được cộng với sum của cột constraint đang xét. Và cái biến x,value_nhan,heso_tmp được làm mới

- Còn nếu không thì em vẫn sẽ tiếp tục đi tìm và thử các giá trị trong HẠNG TỬ HIỆN TẠI (i vẫn giữ nguyên) , giá trị sum không được cộng mà nhân tích lũy theo bậc vào biên value_nhan, kèm theo đó là heso vẫn giữa nguyên
- Và đương nhiên sau khi chọn một giá trị để đệ quy thì sau lượt đệ quy đó ta cần phải khôi phục lại trạng thái ban đầu.

=> Và đó là toàn bộ quá trình em tìm CSP với Đa thức có cộng trừ, chia.

5. Assignment Plan

Members	Work
Nguyễn Huy Thành	Level 1
Nguyễn Nhật Nam	Level 2
Phan Hữu Phước	Level 3
Lê Nguyên Thái	Level 4

6. Experiment

- **Level 1:**

Test Case 1:

Equation: TWO+TWO=FOUR

Output: {'T': 7, 'W': 3, 'O': 4, 'F': 1, 'U': 6, 'R': 8}

Size: 64KB

Time to solve: 0.001s

Test Case 2:

Equation: FOUR+TWO=SIX

Output: No solution

Time to solve: 0.001s

Size: 80KB

Test Case 3:

Equation: EIGHT-FOUR=FOUR

Output: {'E': 1, 'I': 7, 'G': 3, 'H': 0, 'T': 4, 'F': 8, 'O': 6 , 'U': 5, 'R': 2}

Time to solve: 0.001s
Size: 68KB

Test Case 4:

Equation: SIX-THREE=THREE
Output: No solution
Time to solve: 0.001s
Size: 68KB

Test Case 5:

Equation: POINT+ZERO=ENERGY
Output: {'P' : 9, 'O' : 8, 'I' : 5, 'N' : 0, 'T' : 4, 'Z' : 3, 'E' : 1, 'R' : 6, 'G' : 7, 'Y' : 2}
Time to solve: 0.001s
Size: 68KB

Test Case 6:

Equation: SEND+MORE=MONEY
Output: {'S' : 9, 'E' : 5, 'N' : 6, 'D' : 7, 'M' : 1, 'O' : 0, 'R' : 8, 'Y' : 2}
Time to solve: 0.001s
Size: 68KB

Test Case 7:

Equation: USA+USSR=PEACE
Output: {'U' : 9, 'S' : 3, 'A' : 2, 'R' : 8, 'P' : 1, 'E' : 0, 'C' : 7}
Time to solve: 0.001s
Size: 80KB

Test Case 8:

Equation: HOW+MUCH=POWER
Output: {'H' : 7, 'O' : 0, 'W' : 5, 'M' : 9, 'U' : 8, 'C' : 3, 'P' : 1, 'E' : 4, 'R' : 2}
Time to solve: 0.001s
Size: 80KB

- **Level 2**

Test Case 1:

Equation:
SO+MANY+MORE+MEN+SEEM+TO+SAY+THAT+THEY+MAY+SOON+TRY+TO+STAY+AT+HOME+SO+AS+TO+SEE+OR+HEAR+THE+SAME+ONE+MAN+TRY+TO+MEET+THE+TEAM+ON+THE+MOON+AS+HE+HAS+AT+THE+OTHER+TEN=TESTS
Output: {'S' : 3, 'O' : 1, 'M' : 2, 'A' : 7, 'N' : 6, 'Y' : 4, 'R' : 8, 'E' : 0, 'T' : 9, 'H' : 5}
Time to solve: 0.01s
Size: 128KB

Test Case 2:

Equation: NO+NO+GUN=HUNT
Output: {'N' : 8, 'O' : 7, 'G' : 9, 'U' : 0, 'H' : 1, 'T' : 2}
Time to solve: 0.01s
Size: 68KB

Test Case 3:

Equation: HALF+FIFTH+TENTH+TENTH+TENTH=WHOLE
Output: {'H' : 6, 'A' : 7, 'L' : 0, 'F' : 1, 'I' : 4, 'T' : 2, 'E' : 5, 'N' : 3, 'W' : 9, 'O' : 8}
Time to solve: 0.01s
Size: 68KB

Test Case 4:

Equation: SEVEN+NINE-ONE=FIFTEEN
Output: No solution
Time to solve: 0.01s
Size: 68KB

Test Case 5:

Equation: NINE+FIVE-SIX=SEVEN
Output:{ 'N' : 9, 'I' : 4, 'E' : 2, 'F' : 3, 'V' : 8, 'S' : 1, 'X' : 5}
Time to solve: 0.01s
Size: 68KB

- **Level 3**

Test case 1:

Equation: A-(C+(A+B)-(C+D)+(C-(D+E-(A-C-(C+A)))))=A
Output: 'A':9 'C':1 'B':2 'D':3 'E':4

Test case 2:

Equation: ABSS-(CADE+(AVD+BAS)-(DVDA+DAAE)+(CVS-(DSD+EVDA-(AAAB-CDAAR-(CEE+AAC)))))=ABC
Output: No solution

Test case 3:

Equation: SEND+(MORE+MONEY)-OR+DIE=NUOYI
Output: 'S':8 'E':2 'N':6 'D':1 'M':5 'O':3 'R':0 'Y':4 'I':9 'U':7

Test case 4:

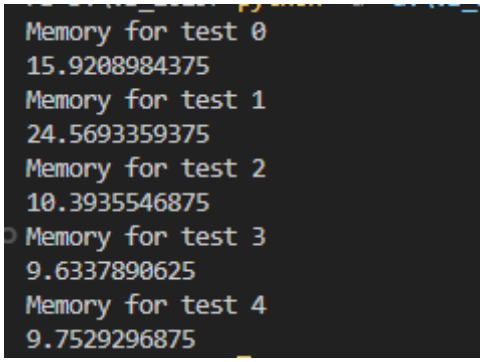
Equation: OUR+YEAR-(YEAH-YARD-YARN)+-YEAH-(YARD-YARN)+DAY-YEAR-YAY+AYYYY-YER+ASS=YERS
Output: No solution

Test case 5:

Equation: SEND+(MORE+MONEY)-OR+DIE=NUOYI
Output: 'S':8 'E':2 'N':6 'D':1 'M':5 'O':3 'R':0 'Y':4 'I':9 'U':7

Total time for 5 tests: 1.0s

Memory for each test cases:



- **Level 4**

Test case 1:

Equation: ABC*CE+AB*(AB-CD*(FCD-A*(-B+CE)))=GFHI
Output: 'A':7 'B':6 'C':2 'E':3 'D':8 'F':1 'G':4 'H':5 'I':0

Test case 2:

Equation: A*A+B*(AB+CE)=FGH
Output: 'A':1 'B':3 'C':5 'E':6 'F':2 'G':0 'H':8

Test case 3:

Equation: -AB*-CD*CC=DE
Output: No solution

Test case 4:
Equation: TWO*TWO=SQUARE
Output: ‘T’ : 8, ‘W’ : 5, ‘O’ : 4, ‘S’ : 7, ‘Q’ : 2, ‘’U’ : 9 , ‘A’ : 3, ‘R’ : 1, ‘E’ : 6

Test case 5:
Equation: (AN*RAN)+BI-(DO*M)=RNIAA
Output: ‘A’ : 9, ‘N’ : 2, ‘R’ : 8, ‘B’ : 7, ‘I’ : 0, ‘D’ : 3, ‘O’ : 5, ‘M’:1

Test case 6:
Equation: (AN*RAU)+CA+BI=KHCK
Output: ‘A’ : 1, ‘N’ : 2, ‘R’ : 6, ‘U’ : 3, ‘C’ : 8, ‘B’ : 5, ‘I’ : 0, ‘K’ : 7, ‘H’ : 4

Test case 7:
Equation: (EM*IU)-ANH-HA=LIKR
Output: ‘E’ : 9, ‘M’ : 0, ‘I’ : 5, ‘U’ : 3, ‘A’ : 1, ‘N’ : 6, ‘H’ : 2, ‘L’ : 4, ‘K’ : 8, ‘R’ : 7

Test case 8:
Equation: I*LOVE+YOU=IIKU
Output: ‘I’ : 5, ‘L’ : 1, ‘O’ : 0, ‘V’ : 3, ‘E’ : 2, ‘Y’ : 4, ‘U’ : 7, ‘K’:6

Test case 9:
Equation: -(A-B*(C-F-(A+B)))*B*C*-(A*C+A)=DEI
Output: 'A':1 'B':6 'C':9 'F':2 'D':4 'E':8 'I':5

Test case 10:
Equation: ABC*(D+E)=FGH
Output: 'A':1 'B':6 'C':3 'D':2 'E':4 'F':9 'G':7 'H':8

Test case 11:
Equation: -ABC*(-CE*(AB+CD))*CD=DE
Output: No solution

Test case 12:
Equation: -AB*-(-CE+AB)*-CD=DE
Output: No solution

Test case 13:

Equation: -AB*-CD*AE*-AB*CC=DE
Output: No solution

Test case 14:

Equation: A*A00+A*A0+A*A0+A*A=ABA
Output: 'A':1 'B':2

Test case 15:

Equation: AA*AA=ABA
Output: ‘A’ : 1 ‘B’ : 2

Test case 16:

Equation: SEND*THE=MONEY
Output: “No solution”

Total time to solve for 16 tests: 0.8s

Memory for each test cases:

```
Memory for test 0
91.0791015625
Memory for test 1
9.5283203125
Memory for test 2
9.5361328125
Memory for test 3
9.7197265625
Memory for test 4
9.5615234375
Memory for test 5
9.7197265625
Memory for test 6
9.7529296875
Memory for test 7
9.5615234375
Memory for test 8
10.8095703125
Memory for test 9
9.5615234375
Memory for test 10
37.5283203125
Memory for test 11
9.8681640625
Memory for test 12
11.4541015625
Memory for test 13
9.3056640625
Memory for test 14
9.2841796875
Memory for test 15
9.2841796875
Memory for test 16
9.3056640625
```

Reflection and comment:

Trong quá trình test, dường như khi nâng cấp biểu thức lên thành một biểu thức khó và dài thì khi chạy chương trình, em không thấy nhiều sự khác biệt giữa các test.

Trong quá trình chạy như biểu thức dưới dây:

```
Nhap bieu thuc hoac da thuc:
(BAEK-GD)*(BCD-(KFGFE+BD)-BF-(IC-KE)*(DC-BC)*(BGDG-KGG)-(BC+CE)-(CC*BBB-(BB-H)))=GBIIICGC
```

Khi chạy thì chương trình mặc định của python với số lượng recursion giới hạn ban đầu không thể giải ra được solution.
Do đó, em nâng giới hạn recursion lên thì kết quả có thể giải ra ngay lập tức.

```
21   # Đặt giới hạn đệ quy mới về 1000000000 để phục vụ cho các test to
22   sys.setrecursionlimit(1000000000)
```

```
Solution la:
B : 1
A : 0
E : 4
K : 9
G : 6
D : 3
C : 2
F : 5
I : 8
H : 7
```

Ngoài ra trong một số trường hợp, khi chạy biểu thức quá dài, lớn và phức tạp như

```
(BADK-GE)*(BCE-(KFGFD+BE)-BF-(IC-KD)*(EC-BC)*(BGEG-KGG)-(BC+CD)-(CC*BBB-(BB-H)))*(EAFA-(CKEC-CKBG))=BIHHGIKIGKAI
```

Lúc này chương trình trên máy tính không thể chạy ra kết quả nhưng khi em chạy trên Google colab thì nó có thể chạy bình thường và giải ra gần như tức thì.

```
Nhap bieu thuc hoac da thuc:
(BADK-GE)*(BCE-(KFGFD+BE)-BF-(IC-KD)*(EC-BC)*(BGEG-KGG)-(BC+CD)-(CC*BBB-(BB-H)))*(EAFA-(CKEC-F))=HGBCCFGCCG
{'(': 1, 'B': 1, 'G': 1, '*': 1, 'K': 1, '-': 1, 'I': 1, 'E': 1, 'C': 1, 'H': 1, ')': 1, 'F': 1, '=': 1}
9
{'(': 1, 'B': 1, 'G': 1, '*': 1, 'K': 1, '-': 1, 'I': 1, 'E': 1, 'C': 1, 'H': 1, ')': 1, 'F': 1, '=': 1}
Solution 1a:
B : 1
A : 0
D : 4
K : 9
G : 6
E : 3
C : 2
F : 5
I : 8
H : 7
```

Theo em vì Google colab có lượng memory lớn và CPU xử lý nên có thể tránh các trường hợp về bộ nhớ hay liên quan đến cấu hình máy tính khi chạy.

7. Usage

- Cần môi trường python (có thể cài đặt bất cứ phiên bản nào từ phiên bản python 3.10.1)
- Input có thể lấy từ các file input.txt ở các level (mặc định thì input sẽ lấy từ file input.txt)
- Sau khi chạy chương trình, chương trình sẽ tạo ra file output.txt chứa kết quả là một dãy các số theo thứ tự của chữ cái alphabet như yêu cầu của đề bài.
- Các kết quả sẽ được ghi tiếp vào file của file output.txt

8. Evaluation

Members	Work	Completion
Nguyễn Huy Thành	Level 1	100%
Nguyễn Nhật Nam	Level 2	100%
Phan Hữu Phước	Level 3	100%
Lê Nguyên Thái	Level 4	100%

9. References

-Slide bài giảng AI của cô Nguyễn Ngọc Thảo