# 程式語言 HW3_Prolog 作業說明

資訊三乙 陳華嚴 F04056154

1. 執行環境：Ubuntu 16.04 Shell
2. 執行步驟：swipl -q –s <檔名>
3. 完成題目：全部
4. 執行結果：

【Question1-GoldBach】

swipl -q -s goldbach.pl

Input=4 時(需手動輸入 4)

```
yan@yan-VirtualBox-for-compiler:~$ swipl -q -s goldbach.pl
Input: 4
Output:
2 2
```

【Question1-GoldBach】

swipl -q -s goldbach.pl

Input=100 時(需手動輸入 100)

```
yan@yan-VirtualBox-for-compiler:~$ swipl -q -s goldbach.pl
Input: 100
Output:
3 97
11 89
17 83
29 71
41 59
47 53
```

【Question2-lca】

swipl -q -s lca.pl

按照作業說明的測資

```
yan@yan-VirtualBox-for-compiler:~$ swipl -q -s lca.pl
|: 6
|: 1 2
|: 1 4
|: 4 5
|: 2 3
|: 4 6
|: 3
|: 3 4
|: 5 6
|: 1 2
Output:
1
4
1
```

【Question3- reachable】

swipl -q -s reachable.pl

按照作業說明的測資

```
yan@yan-VirtualBox-for-compiler:~$ swipl -q -s reachable.pl
|: 6 6
|: 1 2
|: 2 3
|: 3 1
|: 4 5
|: 5 6
|: 6 4
|: 2
|: 1 3
|: 1 5
Output:
Yes
No
```

5. 程式碼說明：

【Question1-GoldBach】

這一大題我有上網查詢質數的寫法，is_prime 原來是利用 2、3 都為質數作為恆真敘述，若 P 大於 3，is_prime 先判斷除以 2 後不能為 0(因為為 0 的話就不是質數了)，再看看所輸入的數字是否有因數(會使用 has_factor)，has_factor 利用的方法是不斷地看看能否除以 3、5、7…(2n+1)的數字(因為 2 的倍數的檢查會再)，最終會執行到 N 的 square root 時就停止(即為數學定理中質數只會存在於小於數值的平方根)，如此就可以知道是否所輸入的數字為質數。

而這一題我使用的演算方法是從 Geekforgeeks 網站找到的：

(1)Find the prime numbers

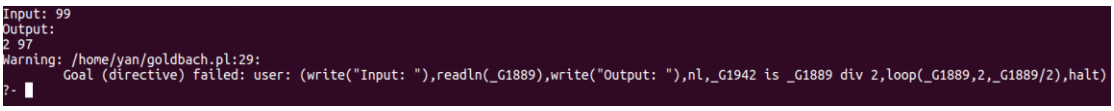
(2) one by one subtract a prime from N and then check if the difference is also a prime, if yes then express it as a sum.

所以我寫出了 goldbach(X, Y) 和 loop(X, Y, K)，前者是利用上述的演算法來決定是否能得出兩個質數並且和為所輸入的數值；後者則為 loop 的功能實現( loop 裡面包著 goldbach，所以每次 loop 都會判斷一次加數是否為 prime )，K 是決定能夠 iteration 幾次的關鍵，於是在":-"就只需要在 K 的位置輸入" 輸入的數值除以 2" ，因為這麼做才可以使得所印出來的兩個數不會再多印一次。如果說不是這麼做的話，那印出來的結果會是如下圖：

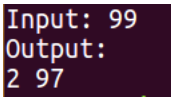
```
Input: 30
Output:
7 23
11 19
13 17
17 13
19 11
23 7
```

就不符合本作業由小到大排列的要求。

另外想特別說明我在這題遇到的困難：一開始我的 loop(I, 2, Q)那裏不是這樣寫的，我是寫 loop(I, 2, I/2)，可是這樣在輸入奇數的時候就會跑出 warning，儘管這樣答案是對的，但看起來就很不舒服…（如下圖）




於是我在猜想應該是因為除以2的關係，所以上網找到了Q is div(I, 2)的寫法，再將 Q 值填寫入 loop 裡，所以實作的時候就使用 loop(I, 2, Q)，再測試 99 的話：




就變成沒有 warning 的令人開心的答案了～:D

【Question2-lca】

直接看到主程式的部分(Line22~26)，會要求您輸入點的數目，再來就會將您輸入的值-1 以當作 relation 的紀錄，所以會利用 assert(parent(A, B))把所輸入的關係都當成正確的資料存入資料庫裡。再者又會讀入要找幾個關係的數值，在來這個數值會搭配空 list 傳入 input_query 裡，當這個數值為 0 的時候結束這個函式，若大於 0 的話則會利用 lca 來尋找，請見到我的 lca(A, B, Output)函式，Output 是用來記錄現在的 lca 是甚麼，因為發現到 prolog 沒有全域變數所以用此法來記錄 lca 是甚麼。再來就不斷地加入 TList 裡，直到 input_query 中的 Q=0 的時候，利用 my_write([First|Rest])印出 list 中所有的值。

【Question3- reachable】

這題跟 lca 那題非常像，都有不斷把結果串在 List 裡，以及建立恆真資料庫(assert)，所以我直接使用了 lca 的一些函式就完成了。相對於 lca，我新增了 all_relation 和 reachable 判斷。all_relation 會 traverse 所有可能的關係，並且加進 TList 裡，再不斷遞迴尋找是否有關係可以串起來。reachable(A, B, Output)會尋找 A~B 點中是否有路徑，如果有的話，Output 會設為”Yes”；沒有的話則會設為”No”，這些 Yes 或 No 也會被加進一個 list 裡，最終再利用 my_write([First|Rest])印出 list 中所有的值。

我覺得 prolog 是很神奇的語言，很多事情都只能透過遞迴來做，因為一不遞迴就會遺失資料(除了設定 assert 的恆真資料庫)，所以要很清楚函式會用到的參數是甚麼、甚麼時候會是 base 條件，但同時也覺得這個語言的先天限制太多其實很可惜，不然他真的是個很特殊的邏輯語言！