Programming language
HW3-PROLOG

Problem1: goldbach's conjecture

Run: swipl -q -s goldbach.pl

Example: (note: input: 0 -> stop the program)

```
~/Desktop/acdemy/programing_language/HW3_F74051051 swipl -q -s goldbach.pl
input: 4
output:
2 2
input: 100
output:
3 97
11 89
17 83
29 71
41 59
47 53
input: 6
output:
3 3
input: 16
output:
3 13
5 11
input: 200
output:
3 197
7 193
19 181
37 163
43 157
61 139
73 127
97 103
input: 2
Please Input One even integer greater than 2
input: 0
```

Explanation:

```
30  %function goldbach
31  two_prime(X1,X2):- prime(X1),prime(X2).
32
33  %% check for the combination from (3,EVEN-3)->(5,EVEN-5)->(7,EVEN-7).....
34
35  % if 2 number are prime, then print it out and check for next conbination  (EVEN,INTEGER+2)
36  % else keeping check for the combination ......->(5,EVEN-5)->(7,EVEN-7).....
37  goldbach(EVEN,INTEGER):-
38      INTEGER =< (EVEN/2),
39      INTEGER2 is (EVEN-INTEGER),
40      prime(INTEGER),prime(INTEGER2),
41      write(INTEGER),
42      write(" "),
43      write(INTEGER2),
44      nl,
45      TMP1 is (INTEGER+2),
46      goldbach(EVEN,TMP1).
47  % else
48  goldbach(EVEN,INTEGER):-
49      INTEGER =< (EVEN/2),
50      INTEGER2 is (EVEN-INTEGER),
51      \+(two_prime(INTEGER,INTEGER2)),
52      TMP2 is (INTEGER+2),
53      TMP2 =< (EVEN/2),
54      goldbach(EVEN,TMP2).
55  % EVEN is no need to check ( range: integer which is < 6 )
56  goldbach(EVEN,INTEGER):-
57      INTEGER > (EVEN/2).
```

```
%function prime
%先過濾掉<=1，2 的倍數，和 3 的倍數,並設<=3 的為 prime
%用 loop 檢查是否為(5 or 7)的倍數, (11 or 13)的倍數, (17 or 19)的倍數, (i or i+2)的倍數...,直到 i*i>要測試的數字,過程如果是的話為false,否則 true

%define prime
prime(2).
prime(3).

prime(N):-
    integer(N),
    N>3,
    N mod 2 =\= 0,
    N mod 3 =\= 0,
    iteration_prime(N,5).

iteration_prime(N,I):-
    I =< sqrt(N),
    N mod I =\= 0,
    N mod (I+2) =\= 0,
    I2 is I +6,
    iteration_prime(N,I2).

iteration_prime(N,I):-
    I > sqrt(N).
```

```
% main
main :-
    repeat,
    write('input: '),
    readln(X),
    (
        (((X mod 2) =:= 0), (X > 2), write('output: '), goldbach(X,3));%如果是二的倍數且大於二，就進入尋找兩個質數相加的function
        (X =:= 4, writeln('2 2'));%如果input is 4,直接output 2 2
        ((((X mod 2) =\= 0); X =:= 2), writeln('Please Input One even integer greater than 2'));% 輸入非偶數，顯示錯誤訊息
        (X =:= 0, halt)% 輸入0，則跳出程式
    ),
    X = 0.
    :-initialization(main).

```

Problem2: lowest common ancestor

Run: swipl -q -s lowest_common_ancestor.pl

Example1:

```
~/Desktop/acdemy/programing_language/HW3_F74051051 swipl -q -s lowest_common_ancestor.pl
|: 6
|: 1 2
|: 2 3
|: 1 4
|: 4 5
|: 4 6
|: 3
|: 3 4
|: 5 6
|: 1 2
1
4
1
?-
```

Example2:

```
~/Desktop/acdemy/programing_language/HW3_F74051051 swipl -q -s lowest_common_ancestor.pl
|: 7
|: 0 1
|: 0 2
|: 1 3
|: 2 4
|: 2 5
|: 5 6
|: 3
|: 3 6
|: 4 6
|: 5 6
0
2
5
?-
```

Explanation:

```
45  % main
46  :-
47      readln(NODENO),                         % read line # of nodes
48      declare_fact(NODENO-1),                 % construct a tree(clarify the fact)
49      readln(QUERYNO),                        % read line # of queries
50      call_lca(QUERYNO, _L1, _L2).            % call lca refer to each query
51
```

```
24  % construct tree refer to esch command
25  declare_fact(NODENO) :-
26      NODENO > 0,                             % check NODENO if bigger than 0
27      readln(X),                              % read line (parent child)
28      nth0(0, X, P), nth0(1, X, C),           % binds index 0 in X to P, and binds index 1 in X to C
29      assert(parent(P, C)),                   % assert the fact
30      TMP1 is NODENO - 1,                     % TMP1 = NODENO - 1
31      declare_fact(TMP1);                     % if(NODENO!=0) declare_fact(NODENO-1);
32      NODENO = 0.                             % if(NODENO==0) return;
```

```
34  % find lowest common ancestor for each query
35  call_lca(QUERYNO, L1, L2) :-
36      QUERYNO > 0,                            % check QUERYNO if bigger than 0
37      readln(X),                              % read line (node node)
38      nth0(0, X, NODE1), nth0(1, X, NODE2),   % binds index 0 in X to NODE1, and binds index 1 in X to NODE2
39      lca(NODE1, NODE2, L1, L2),              % call lca to find lowest common ancestor
40      TMP2 is QUERYNO - 1,                    % TMP2 = QUERYNO - 1
41      call_lca(TMP2, L2, _L3);                %if(QUERYNO!=0) call_lca(TMP2, L2, _L3)
42      QUERYNO = 0,
43      print_ans(L1).                      %if(QUERYNO==0) print_ans(L1)
```

```
18  % find lowest common ancestor, if found append it to L1
19  lca(A, B, L1, L2) :-
20      A==B -> append(L1, A, L2);
21      ancestor(A, B) -> append(L1, A, L2);
22      parent(X, A),lca(X, B, L1, L2).
```

Problem3: reachable

Run: swipl -q -s reachable.pl

Example1:

```
~/Desktop/acdemy/programing_language/HW3_F74051051 swipl -q -s reachable.pl
|: 6 6
|: 1 2
|: 2 3
|: 3 1
|: 4 5
|: 5 6
|: 6 4
|: 3
|: 1 3
|: 1 5
|: 2 3
YES
NO
YES
?-
```

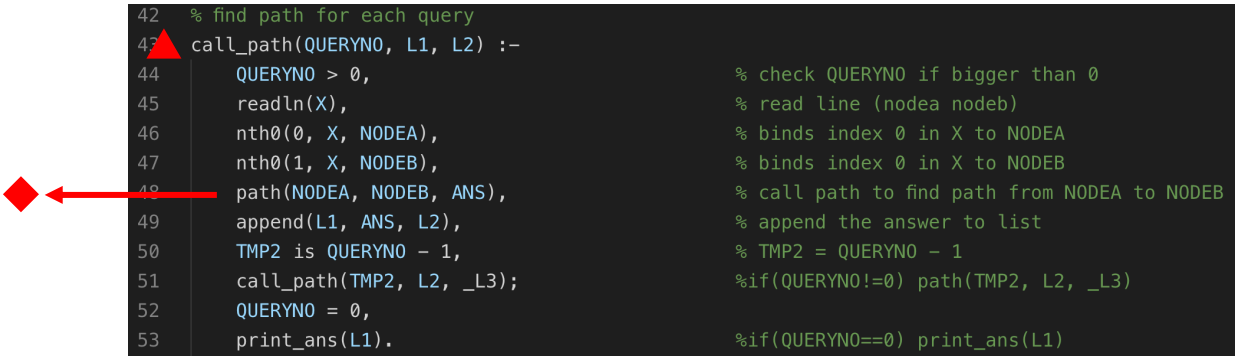
Example2:

```
~/Desktop/acdemy/programing_language/HW3_F74051051 swipl -q -s reachable.pl
|: 7 7
|: 1 2
|: 2 3
|: 3 1
|: 4 5
|: 5 6
|: 6 4
|: 4 7
|: 4
|: 1 3
|: 5 1
|: 7 2
|: 7 6
YES
NO
NO
YES
?-
```

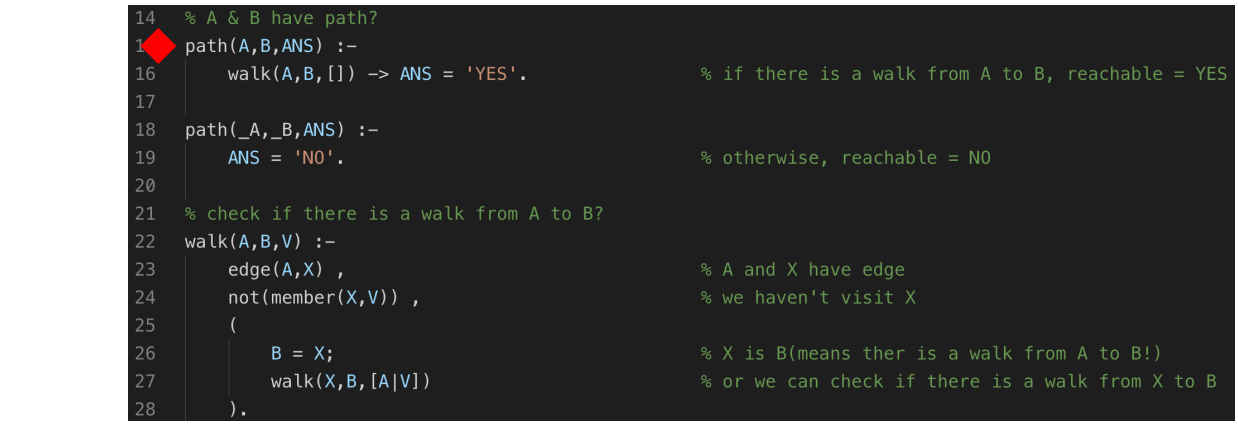
Explanation:

```
55  % main
56  :-
57      readln(X),                          % read line (# of node, # of edge)
58      nth0(0, X, _NODENO),                % binds index 0 in X to NODENO
59      nth0(1, X, EDGENO),                 % binds index 1 in X to EDGENO
60      declare_fact(EDGENO),               % build the graph(clarify the fact)
61      readln(QUERYNO),                    % read line # of queries
62      call_path(QUERYNO, _L1, _L2).       % call path refer to each query
63
```

```
30  % build the graph refer to each command
31  declare_fact(EDGENO) :-
32      EDGENO > 0,                         % check EDGENO if bigger than 0
33      readln(X),                          % read line (node1 node2)
34      nth0(0, X, NODE1),                  % binds index 0 in X to NODE1
35      nth0(1, X, NODE2),                  % binds index 1 in X to NODE2
36      assert(edge(NODE1, NODE2)),         % assert the fact
37      assert(edge(NODE2, NODE1)),         % assert the fact ( undirectly )
38      TMP1 is EDGENO - 1,                 % TMP1 = EDGENO - 1
39      declare_fact(TMP1);                 % if(EDGENO!=0) declare_fact(EDGENO-1);
40      EDGENO = 0.                         % if(EDGENO==0) return;
```



```
% find path for each query
call_path(QUERYNO, L1, L2) :-
    QUERYNO > 0,                          % check QUERYNO if bigger than 0
    readln(X),                            % read line (nodea nodeb)
    nth0(0, X, NODEA),                    % binds index 0 in X to NODEA
    nth0(1, X, NODEB),                    % binds index 0 in X to NODEB
    path(NODEA, NODEB, ANS),              % call path to find path from NODEA to NODEB
    append(L1, ANS, L2),                  % append the answer to list
    TMP2 is QUERYNO - 1,                  % TMP2 = QUERYNO - 1
    call_path(TMP2, L2, _L3);             %if(QUERYNO!=0) path(TMP2, L2, _L3)
    QUERYNO = 0,
    print_ans(L1).                        %if(QUERYNO==0) print_ans(L1)
```



```
% A & B have path?
path(A,B,ANS) :-
    walk(A,B,[]) -> ANS = 'YES'.          % if there is a walk from A to B, reachable = YES

path(_A,_B,ANS) :-
    ANS = 'NO'.                           % otherwise, reachable = NO

% check if there is a walk from A to B?
walk(A,B,V) :-
    edge(A,X) ,                           % A and X have edge
    not(member(X,V)) ,                    % we haven't visit X
    (
        B = X;                            % X is B(means ther is a walk from A to B!)
        walk(X,B,[A|V])                   % or we can check if there is a walk from X to B
    ).
```

append(): append a element to a list

print_ans(): print a list