程式語言 Programming Language HW3

系級: 資訊 109 姓名: 張育豪 學號: E94051209

第一題 Goldbach's conjecture:

執行方式: swipl -q -s goldbach.pl

input:

|: 100 % **輸入一個正偶數**

Output:

3 97

11 89

17 83

29 71

41 59

47 53

程式碼說明(見註解):

```
1  is_prime(2). % Return true.
2  is_prime(3). % Return true.
3  is_prime(P) :-
4      P > 3,
5      P mod 2 =\= 0, % P doesn't have factor "2" .
6      \+ has_factor(P,3). % if P doesn't have odd factor which is >= 3, return true.
7
8  has_factor(N,L) :-
9      N mod L =:= 0. % L is a factor of N.
10 has_factor(N,L) :- % L2 = L + 2, L2 is a factor of N.
11     L * L < N,
12     L2 is L+2,
13     has_factor(N,L2).
14
15 gold(4,[2,2]) :-
16     !. % No backtracking.
17 gold(N, L) :-
18     N mod 2 =:= 0, % N is even number.
19     N > 4
20     -> gold(N, L, 3)
21     ;
22     N =:= 4 % N is 4, special case
23     -> writeln('2 2'),
24        halt.
25 gold(N, [P, Q], P) :-
26     Q is N-P,
27     P < Q,
28     is_prime(Q),
29     write(P), % find a combination , print out to the screen
30     write(" "),
31     writeln(Q),
32     fail.
33
34 gold(N, L, P) :- % Find next combination.
35     P < N,
36     next_prime(P, P1), % P1 is new prime number.
37     gold(N, L, P1). % call back to line 21 predicate.
38 next_prime(P, P1) :- % show P's next valid prime number.
39     P1 is P+2,
40     is_prime(P1),
```

```
    !. % and no backtracking.
next_prime(P, P1) :- % (3,5)->(5,5)->(5,7)->(7,7) keep search possible prime combination.
    P2 is P+2,
    next_prime(P2, P1).
main :- %main function to read a positive even number.
    writeln("input:"),
    readln(Input),
    writeln("Output:"),
    gold(Input, L),
    L
    ; halt.

:-  initialization(main).

```

第二題 LCA:

執行方式: swipl -q -s lca.pl

Input:

```
|: 5                 % 新增五個 edge
|: 1 2               % 以下是 edge 的新增
|: 2 3
|: 1 4
|: 4 5
|: 4 6
|: 3                 % 3 個查詢
|: 3 4               % 查詢 3 與 4 LCA
1                   % output 是 1
|: 5 6               % 查詢 5 與 6 LCA
4                   % output 是 4
|: 1 2               % 查詢 5 與 6 LCA
1                   % output 是 1
```

程式碼說明:

```
ancestor(A, B) :-
    parent(A, B) % A is a parent of B.
    ;            % or
    (
    parent(A, X), % A is a parent of X AND X is a ancestor of B.
    ancestor(X, B)
    ).

lca(A, B) :-
    A == B
    ->  writeln(A) % if A == B , A is lca(A,B).
    ;               % or
    ancestor(A, B) % if A is an ancestor of B,
    ->  writeln(A) % A is lca(A,B).
    ;   parent(X, A), % or X is a parent of A, and
        lca(X, B). % find lca(A's parent X, B).

pop(Z, L) :-
    L=[Z|_]. % Z = the first element of List L.

insert(N) :-
    N>0
    ->  readln(List),
        pop(V1, List),
        last(List, V2), % get the last element of List L.
        assert(parent(V1, V2)), % add a fact, parent(V1,V2).
        insert(N-1) % looping, until N is 0.
    ;
    readln([M|_]), % read a number M to know how many queries the user want.
    query(M). % go go query.

query(M) :-
    M>0
    ->  readln(List2),
        pop(P1, List2),
        last(List2, P2), % get the last element of List L.
        lca(P1, P2), % lca check
        query(M-1) % looping, until M is 0.
    ; halt.

main :-
    writeln("Input:"),
    readln([N|_]), % read a number N to know how many relations to add.
    insert(N).

:- initialization (main).
```

第三題 Reachable:

執行方式: swipl -q -s reachable.pl

input

|: 6 6　　% 以下輸入有 6 個 node, 6 個 edge 的新增

|: 1 2

|: 2 3

|: 3 1

|: 4 5

|: 5 6

|: 6 4

|: 2　　% 2 個查詢

|: 1 3　　% 查詢 1 與 3 是否 Reachable

Yes　　% output 是 Yes

|: 1 5　　% 查詢 1 與 5 是否 Reachable

No　　% output 是 No

程式碼說明（請見註解）:

```
isconnected(A, B) :-
        edge(A, B) -> writeln('Yes')  % if A,B has edge, output Yes.
        ;
        edge(B, A) -> writeln('Yes') % if A,B has edge, output Yes.
        ;
        edge(A, X), % if A,X has edge and X can walk to B, output Yes.
        walk(X, B,[]) -> writeln('Yes')
        ;
        edge(B, X), % if B,X has edge and X can walk to A, output Yes.
        walk(X, A,[]) -> writeln('Yes')
        ; writeln('No'). % there is no path between A and B.

walk(A, B, Path) :-
    edge(A, X),             % if A has edge to X, and
    \+ memberchk(X, Path),  % X is not visited, and
    (B = X                  % (B is X or X can walk to B).
    ;walk(X, B, [A|Path])   % Add A to Path.
    ).

pop(Z, L) :-
    L=[Z|_]. % get the first element of List L.

% insert is the same of lca.pl -> insert(N)
insert(E):-
    E>0
    ->  readln(List),
        pop(V1, List),
        last(List, V2),
        assert(edge(V1, V2)),
        insert(E-1)
    ;
    readln([M|_]),
    query(M).

% query is the same of lca.pl -> query(M)
query(M):-
    M>0
    ->  readln(List2),
        pop(P1, List2),
        last(List2, P2),
        isconnected(P1, P2),
        query(M-1)
    ; halt.
main :-
    writeln('input'),
    readln(L),
    pop(N,L), % read a number N to know how many nodes to add.
    last(L,E), % read a number E to know how many edges to add.
    N > 1
    ->insert(E)
    ; main.

:- initialization (main).
```