Homework 3 - Prolog

SWI-Prolog

Version: 7.6.4

SWI-Prolog Office Website:
http://www.swi-prolog.org/

Install SWI-Prolog on Windows

Go to this website,

http://www.swi-prolog.org/download/stable

32 bit:
SWI-Prolog 7.6.4 for Windows

64 bit:
SWI-Prolog 7.6.4 for Windows 64-bit edition

Install SWI-Prolog on Windows

Add the location of swipl binary file to your PATH variable.

For example, I install swi-prolog in "`C:\Program Files\swipl`".

I should add the path "`C:\Program Files\swipl\bin`" to my PATH variable.

Install SWI-Prolog on Linux

use APT

```
$ sudo apt-get install swi-prolog
```

use YUM

```
$ sudo yum install swi-prolog
```

others

```
Not my business...:)
```

Install SWI-Prolog on OS X

use homebrew

```
$ brew install swi-prolog
```

use macports

```
$ sudo port install swi-prolog
```

If you never believe anyone,

Compile by yourself...

SWI-Prolog source for 7.6.4

Command Line Interface

for Windows

命名提示字元

for Unix-like OS

終端機

Interactive Env

```
$ swipl
```

Interactive Env

```
$ swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version
7.6.4)
Copyright (c) 1990-2018 University of Amsterdam, VU
Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
software,
and you are welcome to redistribute it under certain
conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

```
-? X is 3 + 2 * 7.
```

```
-? X is 3 + 2 * 7.
X = 17.

-?
```

```
-? X is 3 + 2 * 7.
X = 17.

-? wirteln('程式語言最好玩了').
```

```
-? X is 3 + 2 * 7.
X = 17.

-? wirteln('程式語言最好玩了').
```

```
-? X is 3 + 2 * 7.
X = 17.

-? wirteln('程式語言最好玩了').
程式語言最好玩了
true.

-?
```

```
-? X is 3 + 2 * 7.
X = 17.

-? wirteln('程式語言最好玩了').
程式語言最好玩了
true.

-? halt.
```

```
-? X is 3 + 2 * 7.
X = 17.

-? wirteln('程式語言最好玩了').
程式語言最好玩了
true.

-? halt.
% halt
$
```

Prolog File

```
1   % filename: sum.pl
2   sum([],0).
3   sum([H|T],S) :- sum(T,S1), S is S1 + H.
4
5
6
7
8
9
10
11
12
13
14
```

```
$ swipl -q
-? [sum].
```

```
$ swipl -q
-? [sum].
% sum compiled 0.00 sec, 3 clauses
true.

-?
```

```
$ swipl -q
-? [sum].
% sum compiled 0.00 sec, 3 clauses
true.

-? sum([1,2,3,4], X).
```

```
$ swipl -q
-? [sum].
% sum compiled 0.00 sec, 3 clauses
true.

-? sum([1,2,3,4], X).
X = 10.

-?
```

```
$ swipl -q
-? [sum].
% sum compiled 0.00 sec, 3 clauses
true.

-? sum([1,2,3,4], X).
X = 10.

-? halt.
```

```
$ swipl -q
-? [sum].
% sum compiled 0.00 sec, 3 clauses
true.

-? sum([1,2,3,4], X).
X = 10.

-? halt.
% halt
$
```

Script File

```
1   % filename: hanoi.pl
2   move(1, X, Y, _) :-
3         write('Move top disk from '), write(X), write('
4   to '), write(Y), nl.
5
6   move(N, X, Y, Z) :-
7         N > 1, M is N - 1, move(M, X, Z, Y), move(1, X,
8   Y, _), move(M, Z, Y, X).
9
10  main :-
11        move(3, left, right, center), halt.
12
13  :- initialization(main).
14
```

```
$ swipl -q -s hanoi.pl
```

```
$ swipl -q -s hanoi.pl
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right

$
```

作業配分:

Problem 1: 滿分40分

Problem 2: 滿分30分

Problem 3: 滿分30分

Problem 1: Goldbach's conjecture

Every even integer greater than 2 can be expressed as the sum of two primes.

Best-known unsolved problem in number theory.
To this day, the conjecture has been shown to hold up through $\mathbf{4 \times 10^{18}}$.

Input

- One even integer greater than 2.

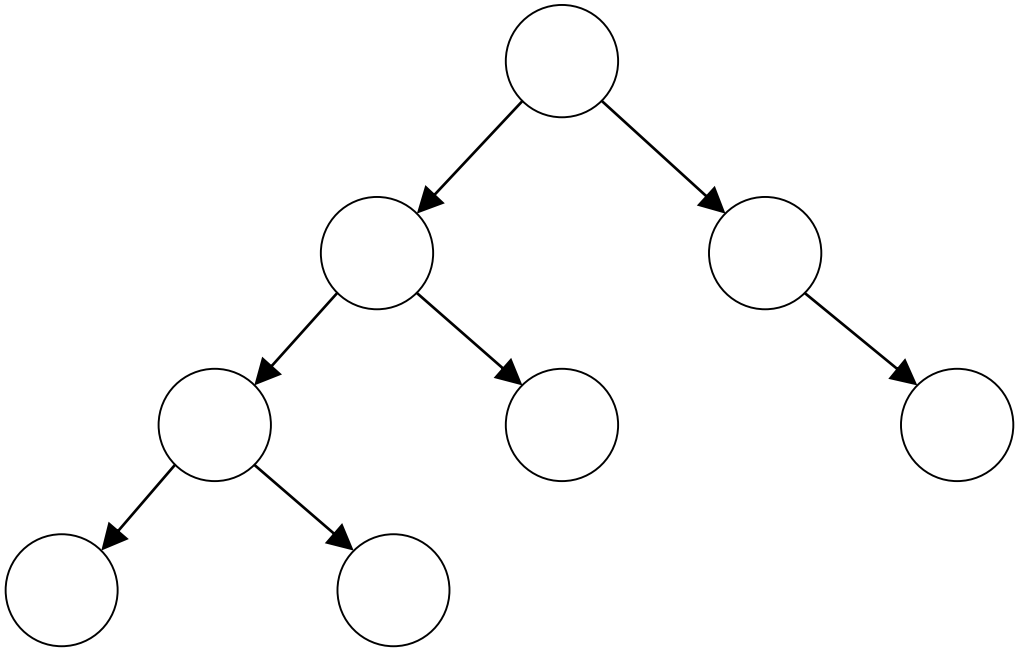
Output

- Two prime integers and their sum is the input integer.
- The output should be ordered.

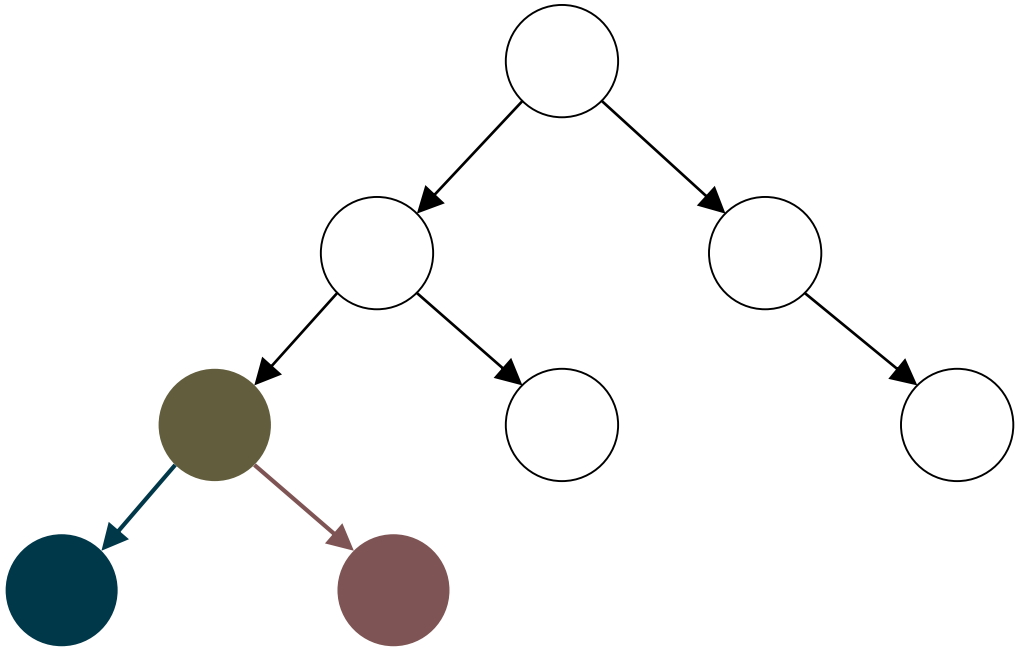
Example

```
Input: 4
Output: 2 2
Input: 100
Output: 47 53
```

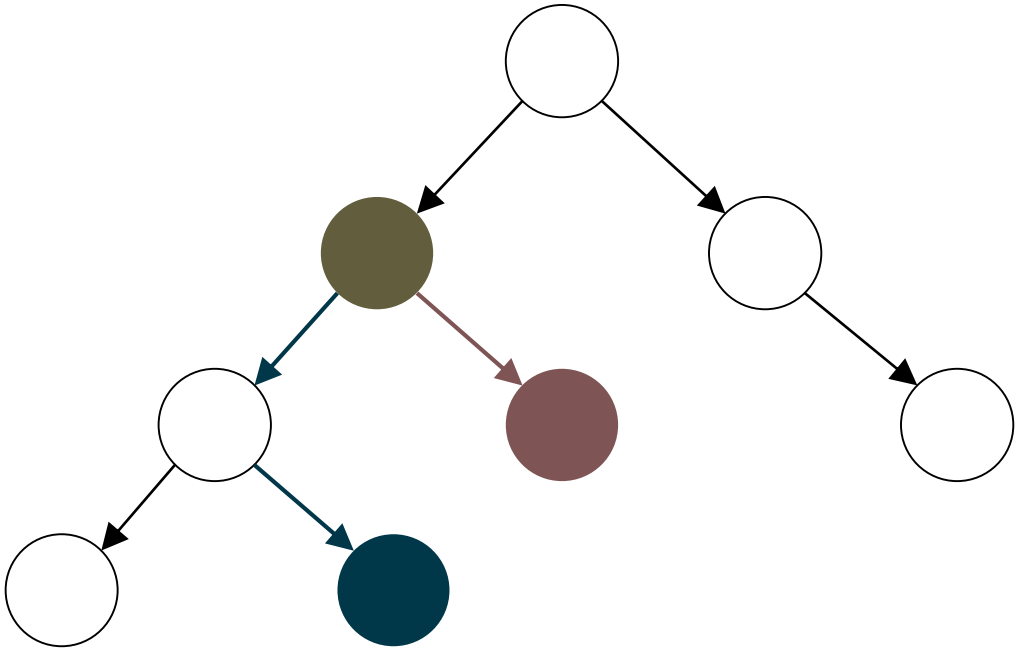
Problem 2 :
Lowest Common Ancestor



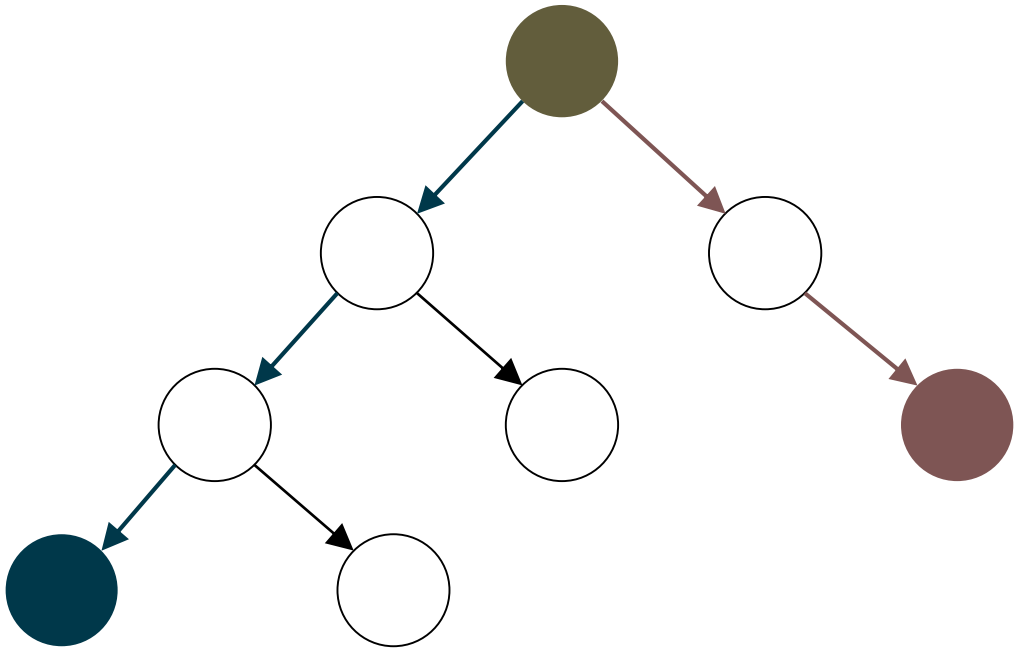
Problem 2 :
Lowest Common Ancestor



Problem 2 :
Lowest Common Ancestor



Problem 2 :
Lowest Common Ancestor



Input

In the first line, use a number N to indicate there are N nodes in the tree.

In the following (N - 1) lines, list the relationship between each pair of two nodes.
For example,“A B”means node A is the parent node of node B.

Then, input a number M in the next line to indicate the numbers of queries.

In the following M lines, list each pair of nodes for qyerying the lowest common ancestor.
For example,“A B”means the query is to find the lowest common ancestor of node A and node B.

Output

The result must be in M lines.
Each line prints the ID of the lowest common ancestor of each query.

Input Format

```
6
1 2
2 3
1 4
4 5
4 6
3
3 4
5 6
1 2
```

Input Format

```
6       -> # of nodes
1 2     -> node 1 is the parent node of node 2
2 3     -> node 2 is the parent node of node 3
1 4     -> node 1 is the parent node of node 4
4 5     -> node 4 is the parent node of node 5
4 6     -> node 4 is the parent node of node 6
3       -> # of queries
3 4     -> Which node is the LCA of node 3 and node 4?
5 6     -> Which node is the LCA of node 5 and node 6?
1 2     -> Which node is the LCA of node 1 and node 2?
```

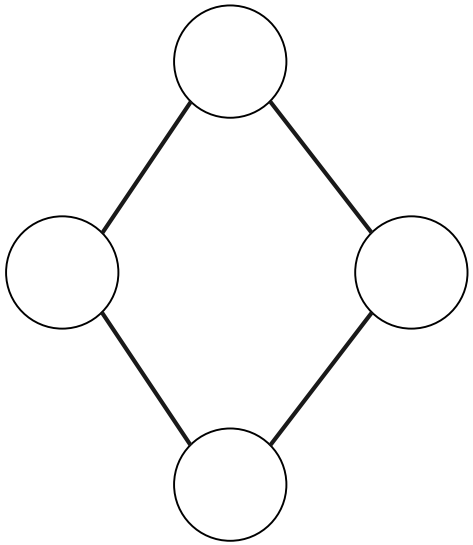
Output Format

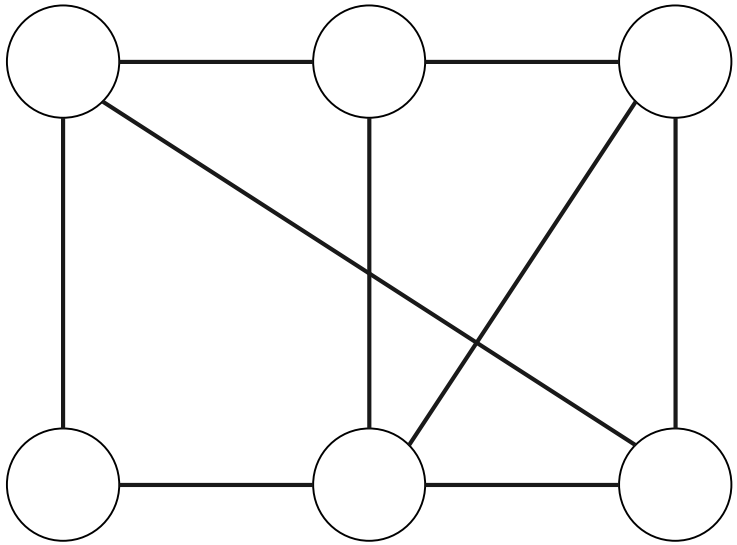
```
1
4
1
```

Output Format

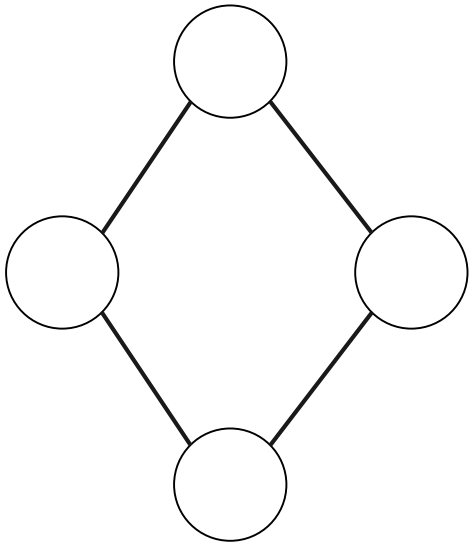
```
1     -> node 1 is the LCA of node 3 and node 4
4     -> node 4 is the LCA of node 5 and node 6
1     -> node 1 is the LCA of node 1 and node 2
```

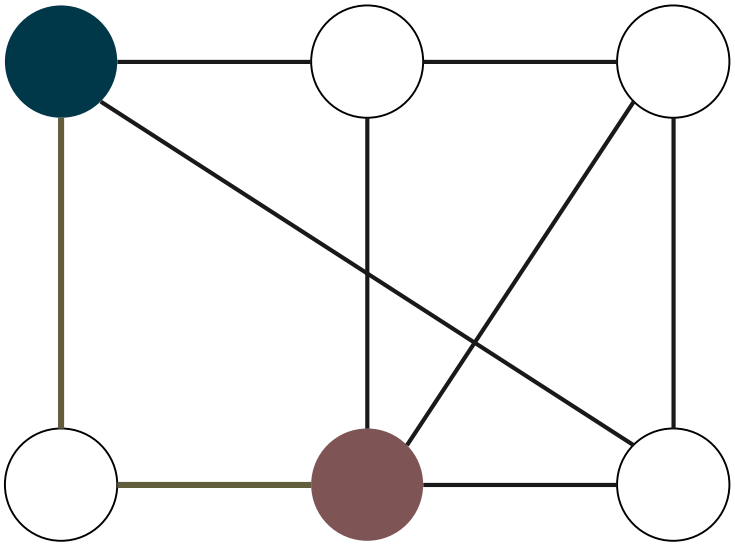
Problem 3 : Reachable



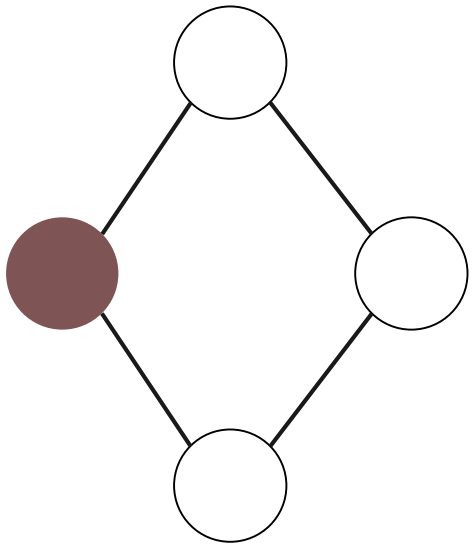


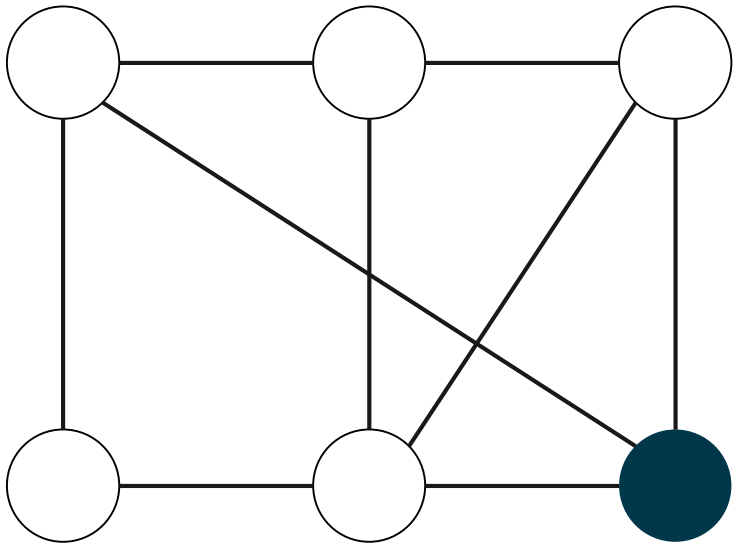
Problem 3 : Reachable





Problem 3 : Reachable





No path to connect the blue node and the red node.

Input

In the first line, use two numbers, N and E, to indicate there are N nodes and E edges in the graph.

In the following E lines, list all paired nodes that have an edge.
For example,“A B” means there is an edge between node A and node B.

Then, input a number M in the next line to indicate the numbers of queries.
- In the following M lines, list each pair of nodes for qyerying the connectivity.
- For example,“A B”means the query is to find the connectivity between node A and node B.

Output

- The result must be in M lines.
- Each line prints “Yes” if node A and node B are connected, or “No” if they are not connected.

Input Format

```
6 6
1 2
2 3
3 1
4 5
5 6
6 1
2
1 3
1 5
```

Input Format

```
6 6     -> # of nodes and # of edges
1 2     -> node 1 and node 2 are connected
2 3     -> node 2 and node 3 are connected
3 1     -> node 3 and node 1 are connected
4 5     -> node 4 and node 5 are connected
5 6     -> node 5 and node 6 are connected
6 1     -> node 6 and node 1 are connected
2       -> # of queries
1 3     -> Are node 1 and node 3 connected?
1 5     -> Are node 1 and node 3 connected?
```

Output Format

```
Yes
No
```

Output Format

```
Yes     -> node 1 and node 3 are connected
No      -> node 1 and node 5 aren’t connected
```

Deadline

2018/05/02 22:00

Any Question?

If you still have any question about SBCL,

Read **T**he **F**riendly **M**anual.

Prolog Tutorial:

http://www.learnprolognow.org/

http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html

SWI-Prolog Document:

http://www.swi-prolog.org/pldoc/refman/

Read **T**he **F**riendly **M**anual or
Use **T**he **F**riendly **G**oogle first,

before you ask teacher or TAs.

Thanks and good luck :)