递归

阶乘运算

```
factorial(0,1).
factorial(A,B) :-
            A > 0,
            C is A-1,
            factorial(C,D),
            B is A*D.

?- factorial(10,What).
What = 3628800.
```

实现外取操作

```
takeout(A,[A|B],B).
takeout(A,[B|C],[B|D]) :-
            takeout(A,C,D).
```

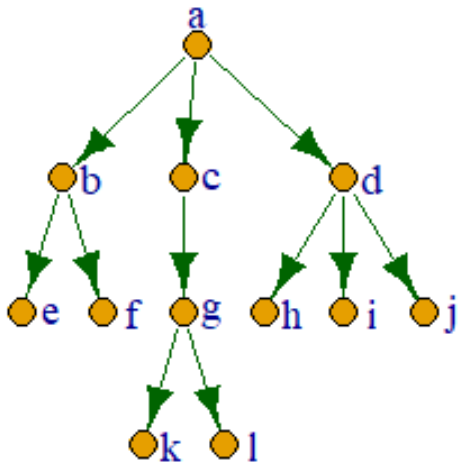
```
?- takeout(X,[1,2,3,4],_), X>3.
X = 4 ;
false.
```

```
?- takeout(X,[1,2,3,4],Y).
X = 1,
Y = [2, 3, 4] ;
X = 2,
Y = [1, 3, 4] ;
X = 3,
Y = [1, 2, 4] ;
X = 4,
Y = [1, 2, 3] ;
false.
```

实现汉诺塔

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

```
?- move(4,left,right,center).
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
Move top disk from left to center
Move top disk from right to left
Move top disk from right to center
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
Move top disk from center to left
Move top disk from right to left
Move top disk from center to right
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
true ;
false.
```



实现树结构

定义几个操作符

```
:- op(500,xfx,'is_parent').
:- op(500,xfx,'is_sibling_of').
:- op(500,xfx,'is_same_level_as').
:- op(500,xfx,'has_depth').
```

创建tree数据库

```
a is_parent b. a is_parent c. a is_parent d.
b is_parent e. b is_parent f.
c is_parent g.
d is_parent h. d is_parent i. d is_parent j.
g is_parent k. g is_parent l.
```

定义“is_sibling_of”

```
X is_sibling_of Y :- Z is_parent X,
    Z is_parent Y,
    X \== Y.
```

定义“is_same_level_as”

```
X is_same_level_as X .
X is_same_level_as Y :- W is_parent X,
    Z is_parent Y,
    W is_same_level_as Z.
```

定义“has_depth”，这里用到了cut技术，即“!”，将在后面介绍。

```
a has_depth 0 :- !.
Node has_depth D :-
    Mother is_parent Node,
    Mother has_depth Dp,
    D is Dp + 1.
```

定义“path”(path是一条从根到该节点的路径)

```
path(Node) :-
    parent_path(Node),
    write(Node).
parent_path(a).
parent_path(Node) :-
    Mother is_parent Node,
    parent_path(Mother),
    write(Mother),
    write(' --> ').
```

定义“height” (height是从该节点到其下某个叶子的最大路径长度)

```
height(N,H) :- setof(Z,ht(N,Z),Set),
        max(Set,0,H).
ht(Node,0) :- leaf(Node), !.
ht(Node,Hn) :- Node is_parent Child,
        ht(Child,Hc),
        Hn is Hc+1.
```

setof是一个内置谓词(参见Predicate setof/3), 搜集各个分支的高度汇总到Set变量中。

定义“leaf”(叶子节点)

```
leaf(Node) :- not(Node is_parent Child).
```

定义“max”

```
max([],M,M).
max([X|R],M,A) :-
    (X > M -> max(R,X,A) ; max(R,M,A)).
```

目标

```
?- max([3,9,8],7,X).
X = 9.

?- max([3,9,11,8],14,X).
X = 14.

?- leaf(f).
true.

?- leaf(g).
false.
```

```
?- X is_sibling_of i.
X = h ;
X = j ;
false.

?- X is_same_level_as c.
X = c ;
X = b ;
X = c ;
X = d ;
false.
```

```
?- k has_depth X.
X = 3.

?- parent_path(k).
a --> c --> g -->
true ;
false.
```

```
?- path(k).
a --> c --> g --> k
true ;
false.

?- height(c,X).
X = 2.

?- height(k,X).
X = 0.
```