```
:- use_module(library(clpfd)).

% functor to sum the values in a list
sum_list([], 0).
sum_list([H | T], Sum) :- sum_list(T, S2), Sum #= H + S2.

% functor to multiply all the values in a list
product_list([], 1).
product_list([H | T], Product) :- product_list(T, S2), Product #= H * S2.

% functor to subtract two values in a list, returns only the true value
sub_list([], 0).
sub_list(VARS, Diff) :-
    nth0(0, VARS, T1),
    nth0(1, VARS, T2),
    (Diff #= T1 - T2; Diff #= T2 - T1).

% functor to divide two values in a list, returns only the true value
div_list([], 0).
div_list(VARS, Quotient) :-
    nth0(0, VARS, T1),
    nth0(1, VARS, T2),
    (Quotient #= T1 // T2; Quotient #= T2 // T1).

% some of my queries I used for testing:
% ---------------------------------------------------------
%check_constraint(cage(id, X, [[0,1]]), [[3, 5 ,2, 6]]).
%check_constraint(cage(id, X, [[0,1]]), [[3, 5], [2, 6]]).
%check_constraint(cage(add, X, [[0,0],[0,1]]), [[3, 5], [2, 6]]).
%check_constraint(cage(mult, X, [[0,0],[0,1]]), [[3, 5], [2, 6]]).
%check_cages([cage(mult, 15, [[0,0],[0,1]]),(cage(id, 6, [[1,1]])], [[3, 5], [2, 6]]).
%---------------------------------------------------------

% cell_values(Cells, Solution, Cell_Values)
% Val must return a list of Values from the list of Coordinates
% S is the solution we search for the value in
cell_values([], S, []).
cell_values([[I, J] | T1], S, Val) :-
    nth0(I, S, Row),
    nth0(J, Row, V),
    Val = [V | T2],
    cell_values(T1, S, T2).

% defined the cage functor that we use to create each cage in the puzzle
cage(id, VALUE, CELLS).

% checks to see if all the values in the specified cells satisfies the target value of the cage
% must add id, mult, div, add, sub
check_constraint(cage(id, VALUE, CELLS), S) :-
    cell_values(CELLS, S, [VALUE]).
check_constraint(cage(add, VALUE, CELLS), S) :-
    cell_values(CELLS, S, VAL),
    sum_list(VAL, VALUE).
check_constraint(cage(mult, VALUE, CELLS), S) :-
    cell_values(CELLS, S, VAL),
    product_list(VAL, VALUE).
check_constraint(cage(sub, VALUE, CELLS), S) :-
    cell_values(CELLS,S, VAL),
    sub_list(VAL, VALUE).
check_constraint(cage(div, VALUE, CELLS), S) :-
    cell_values(CELLS,S,VAL),
    div_list(VAL, VALUE).
```

```
% recursive function to check all of the cages and ensure it returns true
check_cages([], S).
check_cages([ H | T ], S) :-
    check_constraint(H, S),
    check_cages(T, S).




% ALL of the cages we require in the query for the puzzle
/*cages = [cage(mult, 120, [[0,0], [0,1], [1,0], [2,0]]),
         cage(mult, 144, [[0,2], [1,2], [1,3], [1,4]]),
         cage(id, 4, [[0,3]]),
         cage(add, 6, [[0,4], [0,5]]),
         cage(div, 2, [[1,1], [2,1]]),
         cage(div, 3, [[1,5], [2,5]]),
         cage(sub, 3, [[2,2], [3,2]]),
         cage(sub, 4, [[2,3], [2,4]]),
         cage(mult, 15, [[3,0], [3,1]]),
         cage(add, 16, [[3,3], [3,4], [4,3], [5,3]]),
         cage(mult, 48, [[3,5], [4,4], [4,5]]),
         cage(sub, 3, [[4,0], [5,0]]),
         cage(sub, 1, [[4,1], [4,2]]),
         cage(sub, 5, [[5,1], [5,2]]),
         cage(mult, 6, [[5,4], [5,5]])]
 */

% Solve the KenKen Puzzle
solve(CAGES, S) :-
    length(S, 6),
    transpose(S, Cols),
    length(Cols, 6),
    append(S, VALS),
    VALS ins 1..6,
    check_cages(CAGES,S),
    maplist(all_different, S),
    maplist(all_different, Cols),
    maplist(label, S).
```