PP VÀ NL LT

"Programming with Logic"

• Very different from other programming languages

– Declarative (not procedural)

– Recursion (no “for” or “while” loops)

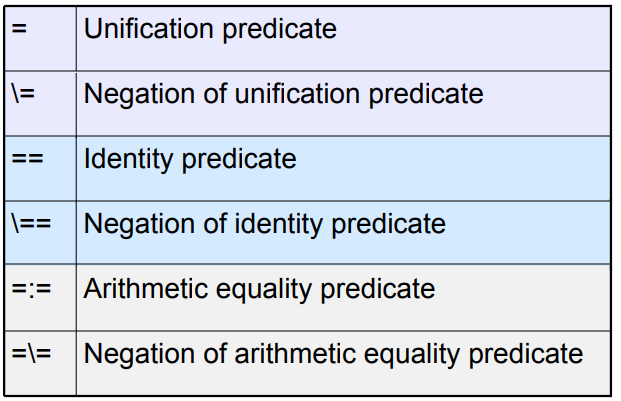
– Relations (no functions)

– Unification

1. Two different **uninstantiated** variables are not identical terms %==/2 trái ngược vs \==/2
2. Variables **instantiated** with a term T are identical to T

?a==a.

yes

?- a==b.

no

?- a=='a'.

yes

?- a==X.

X = \_443

No

?- X==X.

X = \_443

yes

?- Y==X.

Y = \_442

X = \_443

no

?- a=U, a==U.

U = \_443

Yes

?- 2+3 == +(2,3).

yes

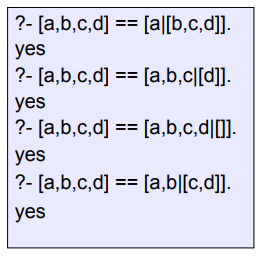
?- -(2,3) == 2-3.

yes

?- (4 <2)== <(4,2)

Yes

?- .(a,[]) == [a]. yes ?- .(f(d,e),[]) == [f(d,e)]. yes ?- .(a,.(b,[])) == [a,b]. yes ?- .(a,.(b,.(f(d,e),[]))) == [a,b,f(d,e)]. Yes ?- X=a, atom(X). X = a yes ?- atom(X), X=a. no



Code:

1. member(X,[X|T]). %tìm kiếm member(X,[yolanda,trudy,vincent,jules]).

member(X,[H|T]):- member(X,T).

1. a2b([],[]). %so sánh độ dài chuỗi a2b([a,t,a,a],[b,b])

a2b([\_|L1],[\_|L2]):- a2b(L1,L2).

1. addThreeAndDouble(X, Y):- Y is (X+3) \* 2.

Note: is(X, +(3,2)).

X = 5 yes

1. len([],0).

len([\_|L],N):- len(L,X), N is X + 1. or

acclen([],Acc,Length):- Length = Acc.

acclen([\_|L],OldAcc,Length):- NewAcc is OldAcc + 1, acclen(L,NewAcc,Length).

1. append([], L, L). %ghép chuỗi

append([H|L1], L2, [H|L3]):- append(L1, L2, L3).

Extra:

prefix(P,L):- append(P,\_,L).

suffix(S,L):- append(\_,S,L).

sublist(Sub,List):- suffix(Suffix,List), prefix(Sub,Suffix).

1. **Các bài toán AI**
2. Đây là một vấn đề cổ điển trong khoa học máy tính. Mục tiêu là đặt tám nữ hoàng trên bàn cờ sao cho không có hai nữ hoàng nào tấn công lẫn nhau; tức là không có hai nữ hoàng trong cùng một hàng, cùng một cột hoặc trên cùng một đường chéo.

Gợi ý: Thể hiện vị trí của các nữ hoàng dưới dạng danh sách các số 1..N. Ví dụ: [4.2,7,3,6,8,5,1] có nghĩa là nữ hoàng ở cột thứ nhất ở hàng 4, nữ hoàng ở cột thứ hai ở hàng 2, v.v ... Sử dụng hàm tạo và mô hình -test.

Chúng tôi khái quát vấn đề ban đầu này bằng cách cho phép một chiều N tùy ý của bàn cờ.

Chúng tôi biểu diễn cho vị trí của các nữ hoàng bằng một danh sách các số 1..N.

Ví dụ: [4.2,7,3,6,8,5,1] có nghĩa là nữ hoàng ở cột thứ nhất ở hàng 4, nữ hoàng ở cột thứ hai ở hàng 2, v.v ... Bằng cách sử dụng hoán vị của các số 1..N chúng tôi đảm bảo rằng không có hai nữ hoàng trong cùng một hàng. Vấn đề duy nhất còn lại phải được thực hiện là vấn đề đường chéo. Một nữ hoàng được đặt ở cột X và hàng Y chiếm hai đường chéo: một trong 2 đường chéo, với số C = X-Y, đi từ dưới cùng bên trái sang trên cùng bên phải, đường còn lại, được đánh số D = X + Y, đi từ trên cùng bên trái sang góc phải ở phía dưới. Trong vị từ kiểm tra, chúng tôi theo dõi các đường chéo đã bị chiếm trong C và D.

1. Trạng thái bài toán:

- Trạng thái ban đầu: 1 quân hậu ở bất kỳ ô nào trên bàn cờ.

- Trạng thái kết thúc: 8 quân hậu đặt trên 8 ô của bàn cờ sao cho không có quân hậu nào có thể “ăn” được quân hậu khác.

1. Phân tích bài toán: N quân hậu sẽ được đặt mỗi con một hàng vì hậu ăn được ngang, ta gọi quân hậu sẽ đặt ở hàng 1 là quân hâu 1, quân hậu ở hàng 2 là quân hậu 2, tương tự với các quân hậu còn lại. Vậy một nghiệm của bài toán sẽ được biết khi ta tìm ra được vị trí cột của những quân hậu. - 1 đường chéo theo hướng ĐB-TN bất kì sẽ đi qua một số ô, các ô đó có tính chất: Hàng +Cột=C (Hằng số). Với mỗi đường chéo ĐB-TN ta có 1 hằng số C và với một hằng số C: 2<= C <=2n xác định duy nhất 1 đường chéo ĐB-TN vì vậy ta có thể đánh chỉ số cho các đường chéo ĐB-TN từ 2 đến 2n.

- 1 đường chéo theo hướng ĐN-TB bất kì sẽ đi qua một số ô, các ô đó có tính chất: Hàng -Cột=C (Hằng số). Với mỗi đường chéo ĐN-TB ta có 1 hằng số C và với một hằng số C: 1-n<= C <= n-1 xác định duy nhất 1 đường chéo ĐN-TB vì vậy ta có thể đánh chỉ số cho các đường chéo ĐN-TB từ 1-n đến n-1.

1. Phương pháp giải:

Ứng dụng thuật toán quay lui (Backtracking) để giải quyết bài toán. - Các bước giải bài toán:

+ Xét tất cả các cột, thử đặt quân hậu 1 vào một cột, với mỗi cách đặt như vậy, xét tất cả các cách đặt quân hậu 2 không bị quân hậu 1 ăn, lại thử 1 cách đặt và xét tiếp các cách đặt quân hậu 3…Mỗi cách đặt được quân hậu n cho ta 1 nghiệm. + Khi chọn vị trí cột j cho quân hậu thứ I, thì ta phải chọn ô(i,j) không bị các quân hậu đặt trước đó ăn, tức là phải chọn cột j còn tự do, đường chéo ĐB-TN(i+j) còn tự do, đường chéo ĐN-TB(i-j) còn tự do. + Nếu (i=n) thì được 1 nghiệm, ngược lại thì gọi đệ quy với quân hậu tiếp i+1

/\* prolog tutorial 2.11 Chess queens challenge puzzle \*/

perm([],[]).

perm([X|Y],Z) :- perm(Y,W), takeout(X,Z,W).

takeout(X,[X|R],R).

takeout(X,[F|R],[F|S]) :- takeout(X,R,S).

solve(P) :-

perm([1,2,3,4,5,6,7,8],P),

combine([1,2,3,4,5,6,7,8],P,S,D),

all\_diff(S),

all\_diff(D).

combine([X1|X],[Y1|Y],[S1|S],[D1|D]) :-

S1 is X1 +Y1,

D1 is X1 - Y1,

combine(X,Y,S,D).

combine([],[],[],[]).

all\_diff([X|Y]) :- \+member(X,Y), all\_diff(Y).

all\_diff([X]).

%?- solve(P).

%P = [5,2,6,1,7,4,8,3] ;

%P = [6,3,5,7,1,4,2,8] ;

%...

%?- setof(P,solve(P),Set), length(Set,L).

%...

%L = 92

Or

% The first version is a simple generate-and-test solution.

% queens\_1(N,Qs) :- Qs is a solution of the N-queens problem

queens\_1(N,Qs) :- range(1,N,Rs), permu(Rs,Qs), test(Qs).

% range(A,B,L) :- L is the list of numbers A..B

range(A,A,[A]).

range(A,B,[A|L]) :- A < B, A1 is A+1, range(A1,B,L).

% permu(Xs,Zs) :- the list Zs is a permutation of the list Xs

permu([],[]).

permu(Qs,[Y|Ys]) :- del(Y,Qs,Rs), permu(Rs,Ys).

del(X,[X|Xs],Xs).

del(X,[Y|Ys],[Y|Zs]) :- del(X,Ys,Zs).

% test(Qs) :- the list Qs represents a non-attacking queens solution

test(Qs) :- test(Qs,1,[],[]).

% test(Qs,X,Cs,Ds) :- the queens in Qs, representing columns X to N,

% are not in conflict with the diagonals Cs and Ds

test([],\_,\_,\_).

test([Y|Ys],X,Cs,Ds) :-

C is X-Y, \+ memberchk(C,Cs),

D is X+Y, \+ memberchk(D,Ds),

X1 is X + 1,

test(Ys,X1,[C|Cs],[D|Ds]).

%--------------------------------------------------------------

% Now, in version 2, the tester is pushed completely inside the

% generator permu.

queens\_2(N,Qs) :- range(1,N,Rs), permu\_test(Rs,Qs,1,[],[]).

permu\_test([],[],\_,\_,\_).

permu\_test(Qs,[Y|Ys],X,Cs,Ds) :-

del(Y,Qs,Rs),

C is X-Y, \+ memberchk(C,Cs),

D is X+Y, \+ memberchk(D,Ds),

X1 is X+1,

permu\_test(Rs,Ys,X1,[C|Cs],[D|Ds]).

n-queens:

n\_queens(N, Qs) :-

length(Qs, N),

Qs ins 1..N,

safe\_queens(Qs).

safe\_queens([]).

safe\_queens([Q|Qs]) :-

safe\_queens(Qs, Q, 1),

safe\_queens(Qs).

safe\_queens([], \_, \_).

safe\_queens([Q|Qs], Q0, D0) :-

Q0 #\= Q,

abs(Q0 - Q) #\= D0,

D1 #= D0 + 1,

safe\_queens(Qs, Q0, D1).

You can then run queries like:

**?- N = 8, n\_queens(N, Qs), labeling([ff], Qs).**

Qs = [1, 5, 8, 6, 3, 7, 2, 4] .

**?- N = 20, n\_queens(N, Qs), labeling([ff], Qs).**

Qs = [1, 3, 5, 14, 17, 4, 16, 7, 12, 18, 15, 19, 6, 10, 20, 11, 8, 2, 13, 9] .

The *n*-th integer in the list denotes the row of the queen that is placed in column *n*.

1. **Mã đi tuần**  
   1. Xây dựng bước đi cho quân mã

Giọi x,y là độ dài bước đi trên các trục Oxy. Một bước đi hợp lệ của quân mã sẽ như sau: |x| + |y| = 3 ( Với x,y > 0).  
Khi đó ở một vị trí bất kì quân mã có có 8 đường có thể di chuyển. Chưa xét đến bước đi đó có hợp lệ hay không.  
Các bước đi đó là: ( -2, -1), ( -2, 1), ( -1, -2), ( -1, 2), ( 1, -2), ( 1, 2), ( 2, -1), ( 2, 1)  
Để đơn giản ta sẽ tạo hay mảng X[], Y[] để chứa các giá trị trên. Với mỗi X[i], Y[i] sẽ là một cách di chuyển của quân mã(0 ≤i≤ 7 ).  
2. Kiểm tra tính hợp lệ của bước đi  
Ta sẽ dùng một mảng hai chiều A[n\*n] để lưu vị trí của từng ô trong bàn cờ. Tất cả mảng đều khởi tạo giá trị là 0( quân mã chưa đi qua).  
Giọi x,y là vị trí hiện tại của quân mã, thì vị trí tiếp theo mà quân mã đi sẽ có dạng x+X[i], y+Y[i]. Một vị trí được gọi là hợp lệ thì sẽ thỏa mãn tính chất sau:

* 0 ≤ x+X[i]≤ n-1.
* 0 ≤ x+X[i]≤ n-1.

Nếu bước đi đó là bước đi đúng thì ta sẽ lưu thứ tự của bước đi đó vào mảng A[ x+X[i], y+Y[i] ].

* Đệ quy: Tại mỗi bước lần lượt cho quân mã thử tất cả các nước đi kế tiếp (tám nước đi kế tiếp). Với mỗi bước đi, kiểm tra xem nếu nước đi hợp lệ (chưa đi qua và ở trong bàn cờ) thì thử đi nước này. Nếu quân mã đã đi qua hết bàn cờ thì xuất kết quả. Ngược lại thì gọi đệ quy tiếp cho vị trí mới thử trên. Lưu ý là mỗi khi vị trí đã đi qua được đánh dấu chính bằng chính thứ tự nước đi trên bàn cờ. Sau khi không thử vị trí này thì phải bỏ đánh dấu để chọn giải pháp khác (trường hợp quay lui).
* Nếu coi các ô của bàn cờ là các đỉnh của đồ thị và các cạnh là nối giữa hai đỉnh tương ứng với hai ô mã giao chân thì dễ thấy rằng hành trình của quân mã cần tìm sẽ là một đường đi Hamilton. Ta có thể xây dựng hành trình bằng thuật toán quay lui kết hợp với***phương pháp duyệt ưu tiên Warnsdorff: Nếu gọi deg(x, y) là số ô kề với ô (x, y) và chưa đi qua (kề ở đây theo nghĩa đỉnh kề chứ không phải là ô kề cạnh) thì từ một ô ta sẽ không thử xét lần lượt các hướng đi có thể, mà ta sẽ ưu tiên thử hướng đi tới ô có deg nhỏ nhất trước.*** Trong trường hợp có tồn tại đường đi, phương pháp này hoạt động với tốc độ tuyệt vời: Với mọi n chẵn trong khoảng từ 6 tới 18, với mọi vị trí ô xuất phát, trung bình thời gian tính từ lúc bắt đầu tới lúc tìm ra một nghiệm < 1 giây. Tuy nhiên trong trường hợp n lẻ, có lúc không tồn tại đường đi, do phải duyệt hết mọi khả năng nên thời gian thực thi lại hết sức tồi tệ. (Có xét ưu tiên như trên hay xét thứ tự như trước kia thì cũng vậy thôi).

% P91 (\*\*) Knight's tour

% Another famous problem is this one: How can a knight jump on an

% NxN chessboard in such a way that it visits every square exactly once?

% knights(N,Knights) :- Knights is a knight's tour on a NxN chessboard

knights(N,Knights) :- M is N\*N-1, knights(N,M,[1/1],Knights).

% closed\_knights(N,Knights) :- Knights is a knight's tour on a NxN

% chessboard which ends at the same square where it begun.

closed\_knights(N,Knights) :-

knights(N,Knights), Knights = [X/Y|\_], jump(N,X/Y,1/1).

% knights(N,M,Visited,Knights) :- the list of squares Visited must be

% extended by M further squares to give the solution Knights of the

% NxN chessboard knight's tour problem.

knights(\_,0,Knights,Knights).

knights(N,M,Visited,Knights) :-

Visited = [X/Y|\_],

jump(N,X/Y,U/V),

\+ memberchk(U/V,Visited),

M1 is M-1,

knights(N,M1,[U/V|Visited],Knights).

% jumps on an NxN chessboard from square A/B to C/D

jump(N,A/B,C/D) :-

jump\_dist(X,Y),

C is A+X, C > 0, C =< N,

D is B+Y, D > 0, D =< N.

% jump distances

jump\_dist(1,2).

jump\_dist(2,1).

jump\_dist(2,-1).

jump\_dist(1,-2).

jump\_dist(-1,-2).

jump\_dist(-2,-1).

jump\_dist(-2,1).

jump\_dist(-1,2).

% A more user-friendly presentation of the results ------------------------

show\_knights(N) :-

get\_time(Time), convert\_time(Time,Tstr),

write('Start: '), write(Tstr), nl, nl,

knights(N,Knights), nl, show(N,Knights).

show(N,Knights) :-

get\_time(Time), convert\_time(Time,Tstr),

write(Tstr), nl, nl,

length(Chessboard,N),

Pred =.. [invlength,N],

checklist(Pred,Chessboard),

fill\_chessboard(Knights,Chessboard,1),

checklist(show\_row,Chessboard),

nl, fail.

invlength(N,L) :- length(L,N).

show\_row([]) :- nl.

show\_row([S|Ss]) :- writef('%3r',[S]), show\_row(Ss).

fill\_chessboard([],\_,\_).

fill\_chessboard([X/Y|Ks],Chessboard,K) :-

nth1(Y,Chessboard,Row),

nth1(X,Row,K),

K1 is K+1,

fill\_chessboard(Ks,Chessboard,K1).

% --------------------------------------------------------------------------

1. **An arithmetic puzzle**

% P93 (\*\*\*) Arithmetic puzzle: Given a list of integer numbers,

% find a correct way of inserting arithmetic signs such that

% the result is a correct equation. The idea to the problem

% is from Roland Beuret. Thanx.

% Example: With the list of numbers [2,3,5,7,11] we can form the

% equations 2-3+5+7 = 11 or 2 = (3\*5+7)/11 (and ten others!).

% equation(L,LT,RT) :- L is the list of numbers which are the leaves

% in the arithmetic terms LT and RT - from left to right. The

% arithmetic evaluation yields the same result for LT and RT.

equation(L,LT,RT) :-

split(L,LL,RL), % decompose the list L

term(LL,LT), % construct the left term

term(RL,RT), % construct the right term

LT =:= RT. % evaluate and compare the terms

% term(L,T) :- L is the list of numbers which are the leaves in

% the arithmetic term T - from left to right.

term([X],X). % a number is a term in itself

% term([X],-X). % unary minus

term(L,T) :- % general case: binary term

split(L,LL,RL), % decompose the list L

term(LL,LT), % construct the left term

term(RL,RT), % construct the right term

binterm(LT,RT,T). % construct combined binary term

% binterm(LT,RT,T) :- T is a combined binary term constructed from

% left-hand term LT and right-hand term RT

binterm(LT,RT,LT+RT).

binterm(LT,RT,LT-RT).

binterm(LT,RT,LT\*RT).

binterm(LT,RT,LT/RT) :- RT =\= 0. % avoid division by zero

% split(L,L1,L2) :- split the list L into non-empty parts L1 and L2

% such that their concatenation is L

split(L,L1,L2) :- append(L1,L2,L), L1 = [\_|\_], L2 = [\_|\_].

% do(L) :- find all solutions to the problem as given by the list of

% numbers L, and print them out, one solution per line.

do(L) :-

equation(L,LT,RT),

writef('%w = %w\n',[LT,RT]),

fail.

do(\_).

% Try the following goal: ?- do([2,3,5,7,11]).

1. **dãy số nhị phân có độ dài cho trc**

% (\*\*) P49 Gray codes

% gray(N,C) :- C is the N-bit Gray code

gray(1,['0','1']).

gray(N,C) :- N > 1, N1 is N-1,

gray(N1,C1), reverse(C1,C2),

prepend('0',C1,C1P),

prepend('1',C2,C2P),

append(C1P,C2P,C).

prepend(\_,[],[]) :- !.

prepend(X,[C|Cs],[CP|CPs]) :- atom\_concat(X,C,CP), prepend(X,Cs,CPs).

% This gives a nice example for the result caching technique:

:- dynamic gray\_c/2.

gray\_c(1,['0','1']) :- !.

gray\_c(N,C) :- N > 1, N1 is N-1,

gray\_c(N1,C1), reverse(C1,C2),

prepend('0',C1,C1P),

prepend('1',C2,C2P),

append(C1P,C2P,C),

asserta((gray\_c(N,C) :- !)).

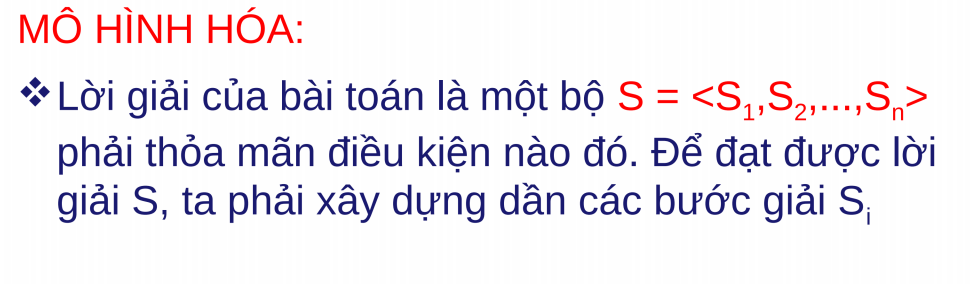
% Try the following goal sequence and see what happens:

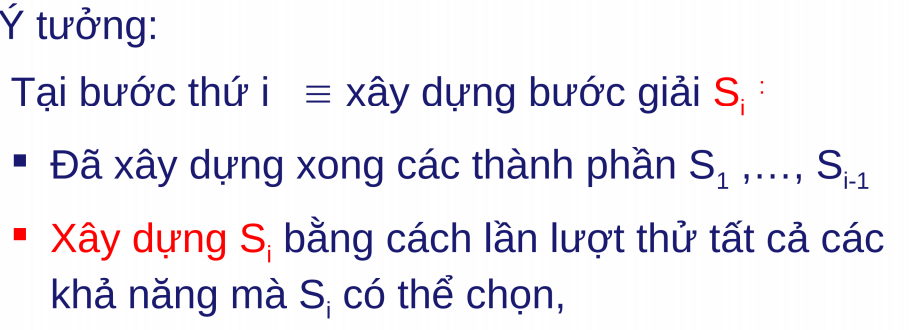
% ?- gray\_c(5,C).

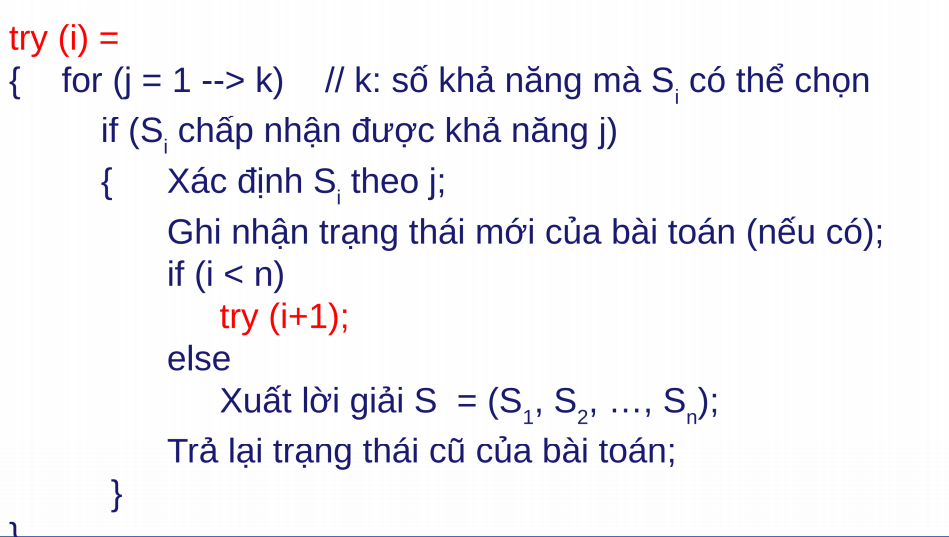
Ý tưởng:

Trong một dãy nhị phân, tại một vị trí mỗi phần tử có thể nhận một trong hai giá trị 0 và 1. Vì một vị trí có hai khả năng xảy ra và có tất cả n vị trí (n là độ dài một dãy nhị phân) nên sẽ có tất cả 2^n (2 mũ n) dãy nhị phân.  
  
Tưởng tượng như sau : - Vị trí đầu tiên có 2 cách chọn (0 và 1).  
                                       - Ứng với mỗi cách chọn vị trí đầu tiên, ta lại có 2 cách chọn cho vị trí thứ hai. (cũng là 0 và 1)  
                                       ...  
  
nên ta sẽ có :   2 \* 2 \* 2 \* ..... \* 2 = 2^n dãy nhị phân.  
                              {có n số 2}  
  
Và cách sử dụng thuật toán quay lui để giải quyết bài toán này cũng y như đoạn "tưởng tượng" trên.  
  
Ta gọi dãy nhị phân là X, và X1, X2, X3,.... Xn là các phần tử từ trái qua phải trong dãy.  
  
Tại vị trí X*i* (1 <= *i* <= n), ta gán cho nó các giá trị mà nó có thể nhận được (ở bài toán này là 0 và 1). Tại mỗi lần gán giá trị cho X*i* ta lại gán các giá trị có thể nhận được (là 0 và 1) cho X*(i+1)*,... cứ thế cho hết độ dài của dãy.

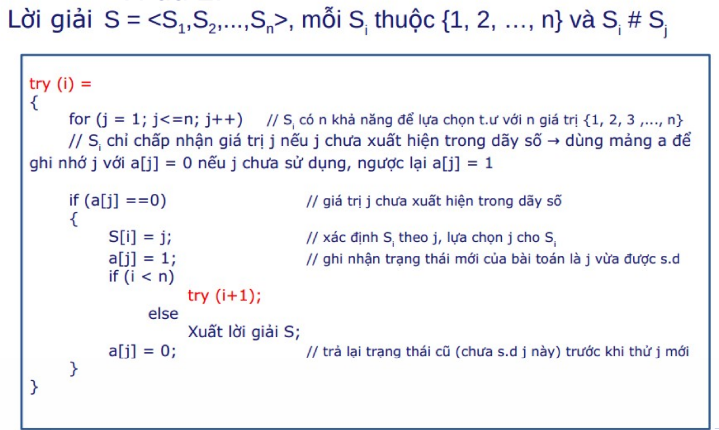
***Phương pháp quay lui***







1. **Liệt kê các hoán vị của n số nguyên dương đầu tiên:**



eval([],\_).

eval([H|T],Set):-member(H,Set),eval(T,Set).

myPermutation(N, Set, L):-length(L,N), eval(L,Set).

1. **Generate the combinations of K distinct objects chosen from the N elements of a list (backtracking)**

% P26 (\*\*): Generate the combinations of k distinct objects

% chosen from the n elements of a list.

% combination(K,L,C) :- C is a list of K distinct elements

% chosen from the list L

combination(0,\_,[]).

combination(K,L,[X|Xs]) :- K > 0,

el(X,L,R), K1 is K-1, combination(K1,R,Xs).

% Find out what the following predicate el/3 exactly does.

el(X,[X|L],L).

el(X,[\_|L],R) :- el(X,L,R).

1. **Bài toán các dãy con có tổng cho trước**

Ý tưởng của thuật toán quay lui dựa trên quan sát sau: xét một phần tử x∈Xx∈X, tồn tại một dãy con có tổng bằng TT nếu một trong hai điều kiện sau là đúng:

1. Tồn tại một tập con của X∖{x}X∖{x} có tổng bằng T−xT−x
2. Tồn tại một tập con của X∖{x}X∖{x} có tổng bằng TT

Giả mã như sau:

SubsetSum(X[1,2,…,n],r,TX[1,2,…,n],r,T):   
    **if** T=0T=0  
        return True  
    **else if** TT > 00 or n==0n==0  
        return False  
    return SubsetSum(X[1,2,…,n−1],T)(X[1,2,…,n−1],T) Or  
                SubsetSum(X[1,2,…,n−1],T−X[n])(X[1,2,…,n−1],T−X[n])

hai lần trên mảng con của XX với kích thước nhỏ hơn 1. Ta có:

T(n)=2T(n−1)+O(1)=O(2n)

1. **Sudoku**

Trong bài này mình sẽ giới thiệu cách giải sudoku bằng thuật toán quay lui. Ý tưởng của thuật toán cũng giống bài toán nn quân hậu. Mỗi bước tìm tập các giá trị khả dĩ để điền vào ô trống, và sau đó đệ quy để điền ô tiếp theo. Giả mã của thuật toán (ở đây chú ý mảng chỉ có kích thước 9×99×9). Thủ tục Feasible(S,x,y,k)(S,x,y,k) kiểm trả xem giá trị kk có khả dĩ với ô S[x][y]S[x][y] không.

Sudoku(S[1,2,…,9][1,2,…,9],x,yS[1,2,…,9][1,2,…,9],x,y):   
    **if** y=10y=10  
        **if** x=9x=9  
            print SS  
       **else**  
            Sudoku(S[1,2,…,9][1,2,…,9],x+1,1S[1,2,…,9][1,2,…,9],x+1,1)  
    **else if** S[x,y]=∅S[x,y]=∅  
        **for**k←1k←1 to 99  
            **if**Feasible(S,x,y,k)(S,x,y,k)  
                S[x,y]←kS[x,y]←k  
                Sudoku(S[1,2,…,9][1,2,…,9],x,y+1S[1,2,…,9][1,2,…,9],x,y+1)  
                S[x,y]←∅S[x,y]←∅        ≪≪ for next branching ≫≫  
    **else**                                    ≪S[x,y]≪S[x,y] is given ≫≫  
        Sudoku(S[1,2,…,9][1,2,…,9],x,y+1S[1,2,…,9][1,2,…,9],x,y+1)

Giả mã của thủ tục Feasible(S,x,y,k)(S,x,y,k) như sau:

Feasible(S[1,2,…,9][1,2,…,9],x,y,kS[1,2,…,9][1,2,…,9],x,y,k):   
    **for** i←1i←1 to 99  
        **if** S[x,i]=kS[x,i]=k  
            return False  
    **for** i←1i←1 to 99  
        **if** S[i,y]=kS[i,y]=k  
            return False  
    a←⌊(x−1)/3⌋,b←⌊(y−1)/3⌋a←⌊(x−1)/3⌋,b←⌊(y−1)/3⌋  
    **for** i←3a+1i←3a+1 to 3a+33a+3  
        **for** j←3b+1j←3b+1 to 3b+33b+3  
           **if** S[i,j]=kS[i,j]=k  
                return False  
    return True