THUẬT TOÁN QUAY LUI

Thuật toán quay lui dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng.

Giả thiết cấu hình cần liệt kê có dạng (x_1, x_2,..., x_n). Khi đó thuật toán quay lui thực hiện qua các bước sau:

1) Xét tất cả các giá trị x_1 có thể nhận, thử cho x_1 nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x_1 ta sẽ:

2) Xét tất cả các giá trị x_2 có thể nhận, lại thử cho x_2 nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x_2 lại xét tiếp các khả năng chọn x_3 ... cứ tiếp tục như vậy đến bước:

n) Xét tất cả các giá trị x_n có thể nhận, thử cho x_n nhận lần lượt các giá trị đó, thông báo cấu hình tìm được (x_1, x_2, ..., x_n).

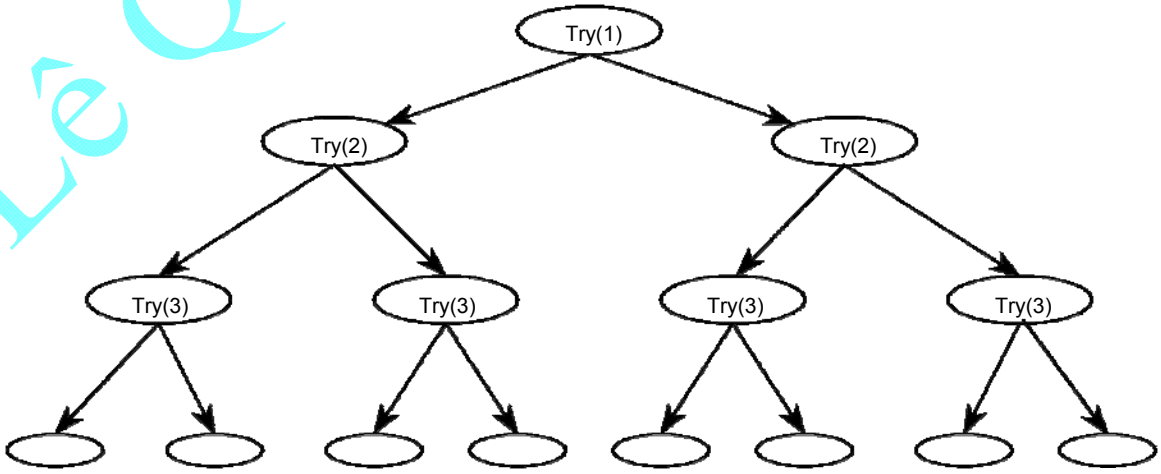
Trên phương diện quy nạp, có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng (x_1, x_2, .., x_n) bằng cách thử cho x_1 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_1 lại liệt kê tiếp cấu hình n - 1 phần tử (x_2, x_3, ..., x_n).

Mô hình của thuật toán quay lui có thể mô tả như sau:

```
{Thủ tục này thử cho xi nhận lần lượt các giá trị mà nó có thể nhận}
procedure Try(i: Integer);
begin
  for (mọi giá trị V có thể gán cho xi) do
    begin
      <Thử cho xi := V>;
      if (xi là phần tử cuối cùng trong cấu hình) then
            <Thông báo cấu hình tìm được>
      else
        begin
          <Ghi nhận việc cho xi nhận giá trị V (Nếu cần)>;
          Try(i + 1); {Gọi đệ quy để chọn tiếp xi+1}
          <Nếu cần, bỏ ghi nhận việc thử xi := V, để thử giá trị khác>;
        end;
    end;
end;
```

Thuật toán quay lui sẽ bắt đầu bằng lời gọi Try(1)

Ta có thể trình bày quá trình tìm kiếm lời giải của thuật toán quay lui bằng cây sau:



Hình 1: Cây tìm kiếm quay lui

I. LIỆT KÊ CÁC DÃY NHỊ PHÂN ĐỘ DÀI N

Biểu diễn dãy nhị phân độ dài N dưới dạng (x_1, x_2, ..., x_n). Ta sẽ liệt kê các dãy này bằng cách thử dùng các giá trị {0, 1} gán cho x_i. Với mỗi giá trị thử gán cho x_i lại thử các giá trị có thể gán cho x_{i+1}.Chương trình liệt kê bằng thuật toán quay lui có thể viết:

*** Thuật toán quay lui liệt kê các dãy nhị phân độ dài n program**

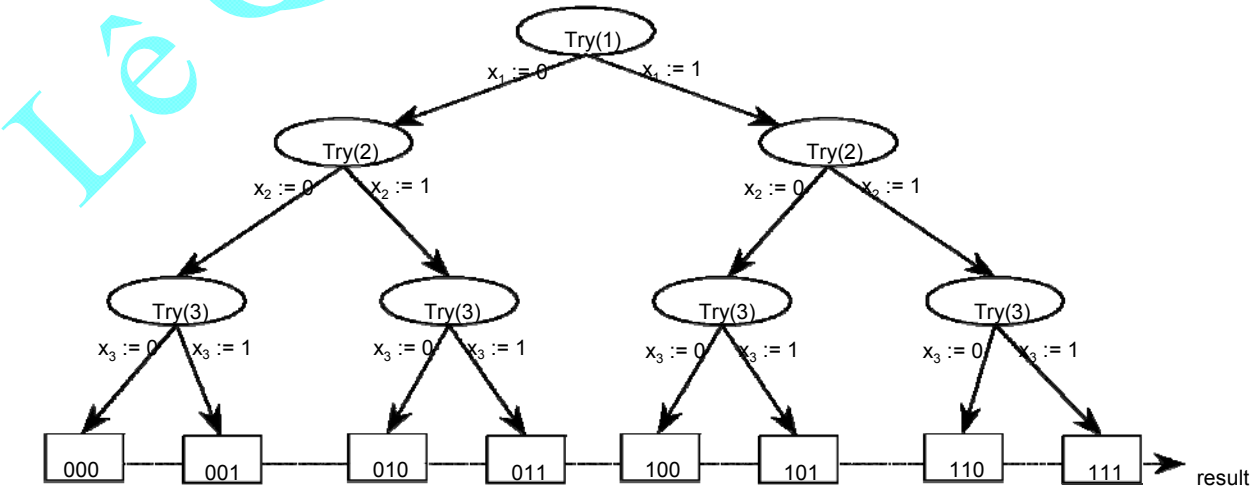
```
BinaryStrings;
const
  max = 30;
var
  x: array[1..max] of Integer;
  n: Integer;

procedure PrintResult;          {In cấu hình tìm được, do thủ tục tìm đệ quy Try gọi khi tìm ra một cấu hình}
var
  i: Integer;
begin
  for i := 1 to n do Write(x[i]);
  WriteLn;
end;

procedure Try(i: Integer);      {Thử các cách chọn xi}
var
  j: Integer;
begin
  for j := 0 to 1 do            {Xét các giá trị có thể gán cho xi, với mỗi giá trị đó}
    begin
      x[i] := j;                {Thử đặt xi}
      if i = n then PrintResult {Nếu i = n thì in kết quả}
      else Try(i + 1);          {Nếu i chưa phải là phần tử cuối thì tìm tiếp xi+1}
    end;
end;

begin
  Assign(Input, 'BSTR.INP'); Reset(Input);
  Assign(Output, 'BSTR.OUT'); Rewrite(Output);
  ReadLn(n);                    {Nhập dữ liệu}
  Try(1);                       {Thử các cách chọn giá trị x1}
  Close(Input);
  Close(Output);
end.
```

Ví dụ: Khi n = 3, cây tìm kiếm quay lui như sau:



Hình 2: Cây tìm kiếm quay lui trong bài toán liệt kê dãy nhị phân

II. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ

Để liệt kê các tập con k phần tử của tập S = {1, 2, ..., n} ta có thể đưa về liệt kê các cấu hình (x_1, x_2, ..., x_k) ở đây các $x_i \in S$ và $x_1 < x_2 < ... < x_k$. Ta có nhận xét:

- $x_k \le n$
- $x_{k-1} \le x_k - 1 \le n - 1$
- ...
- $x_i \le n - k + i$
- ...
- $x_1 \le n - k + 1$.

Từ đó suy ra $x_{i-1} + 1 \le x_i \le n - k + i$ ($1 \le i \le k$) ở đây ta giả thiết có thêm một số $x_0 = 0$ khi xét i = 1. Như vậy ta sẽ xét tất cả các cách chọn x_1 từ 1 ($=x_0 + 1$) đến n - k + 1, với mỗi giá trị đó, xét tiếp tất cả các cách chọn x_2 từ $x_1 + 1$ đến n - k + 2,... cứ như vậy khi chọn được đến x_k thì ta có một cấu hình cần liệt kê. Chương trình liệt kê bằng thuật toán quay lui như sau:

*** Thuật toán quay lui liệt kê các tập con k phần tử**

```
program Combinations;
const
  max = 30;
var
  x: array[0..max] of Integer;
  n, k: Integer;

procedure PrintResult;  (*In ra tập con {x1, x2, ..., xk}*)
var
  i: Integer;
begin
  Write('{');
  for i := 1 to k - 1 do Write(x[i], ', ');
  WriteLn(x[k], '}');
end;

procedure Try(i: Integer);  {Thử các cách chọn giá trị cho x[i]}
var
  j: Integer;
begin
  for j := x[i - 1] + 1 to n - k + i do
    begin
      x[i] := j;
      if i = k then PrintResult
      else Try(i + 1);
    end;
end;

begin
  Assign(Input, 'SUBSET.INP'); Reset(Input);
  Assign(Output, 'SUBSET.OUT'); Rewrite(Output);
  ReadLn(n, k);
  x[0] := 0;
  Try(1);
  Close(Input); Close(Output);
end.
```

Nếu để ý chương trình trên và chương trình liệt kê dãy nhị phân độ dài n, ta thấy về cơ bản chúng chỉ khác nhau ở thủ tục Try(i) - chọn thử các giá trị cho x_i, ở chương trình liệt kê dãy nhị phân ta thử chọn các giá trị 0 hoặc 1 còn ở chương trình liệt kê các tập con k phần tử ta thử chọn x_i là một trong các giá trị nguyên từ $x_{i-1} + 1$ đến $n - k + i$. Qua đó ta có thể thấy tính phổ dụng của thuật toán quay lui: mô hình cài đặt có thể thích hợp cho nhiều bài toán, khác với phương pháp sinh tuần tự, với mỗi bài toán lại phải có một thuật toán sinh kế tiếp riêng làm cho việc cài đặt mỗi bài một khác, bên cạnh đó, không phải thuật toán sinh kế tiếp nào cũng dễ cài đặt.

III. LIỆT KÊ CÁC CHỈNH HỢP KHÔNG LẶP CHẬP K

Để liệt kê các chỉnh hợp không lặp chập k của tập S = {1, 2, ..., n} ta có thể đưa về liệt kê các cấu hình $(x_1, x_2, ..., x_k)$ ở đây các $x_i \in S$ và khác nhau đôi một.

Như vậy thủ tục Try(i) - xét tất cả các khả năng chọn x_i - sẽ thử hết các giá trị từ 1 đến n, mà các giá trị này chưa bị các phần tử đứng trước chọn. Muốn xem các giá trị nào chưa được chọn ta sử dụng kỹ thuật dùng mảng đánh dấu:

- Khởi tạo một mảng $c_1, c_2, ..., c_n$ mang kiểu logic. Ở đây c_i cho biết giá trị i có còn tự do hay đã bị chọn rồi. Ban đầu khởi tạo tất cả các phần tử mảng c là TRUE có nghĩa là các phần tử từ 1 đến n đều tự do.
- Tại bước chọn các giá trị có thể của x_i ta chỉ xét những giá trị j có c_j = TRUE có nghĩa là **chỉ chọn những giá trị tự do.**
- Trước khi gọi đệ quy tìm x_{i+1}: ta đặt giá trị j vừa gán cho x_i là **đã bị chọn** có nghĩa là đặt c_j := FALSE để các thủ tục Try(i + 1), Try(i + 2)... gọi sau này không chọn phải giá trị j đó nữa
- Sau khi gọi đệ quy tìm x_{i+1}: có nghĩa là sắp tới ta sẽ thử gán một **giá trị khác** cho x_i thì ta sẽ đặt giá trị j vừa thử đó thành **tự do** (c_j := TRUE), bởi khi x_i đã nhận một giá trị khác rồi thì các phần tử đứng sau: x_{i+1}, x_{i+2} ... hoàn toàn có thể nhận lại giá trị j đó. Điều này hoàn toàn hợp lý trong phép xây dựng chỉnh hợp không lặp: x_1 có n cách chọn, x_2 có n - 1 cách chọn, ...Lưu ý rằng khi thủ tục Try(i) có i = k thì ta không cần phải đánh dấu gì cả vì tiếp theo chỉ có in kết quả chứ không cần phải chọn thêm phần tử nào nữa.

Input: file văn bản ARRANGES.INP chứa hai số nguyên dương n, k ($1 \le k \le n \le 20$) cách nhau ít nhất một dấu cách

Output: file văn bản ARRANGES.OUT ghi các chỉnh hợp không lặp chập k của tập {1, 2, ..., n}

ARRANGES.INP	ARRANGES.OUT
3 2	1 2 1 3 2 1 2 3 3 1 3 2

*** Thuật toán quay lui liệt kê các chỉnh hợp không lặp chập k**

```
program Arranges;
const
  max = 20;
var
  x: array[1..max] of Integer;
  c: array[1..max] of Boolean;
  n, k: Integer;

procedure PrintResult;      {Thủ tục in cấu hình tìm được}
```

```
var
  i: Integer;
begin
  for i := 1 to k do Write(x[i],' ');
  WriteLn;
end;

procedure Try(i: Integer); {Thử các cách chọn xi}
var
  j: Integer;
begin
  for j := 1 to n do
    if c[j] then    {Chỉ xét những giá trị j còn tự do}
      begin
        x[i] := j;
        if i = k then PrintResult {Nếu đã chọn được đến xk thì chỉ việc in kết quả}
        else
          begin
            c[j] := False;    {Đánh dấu: j đã bị chọn}
              Try(i + 1);  {Thủ tục này chỉ xét những giá trị còn tự do gán cho xi+1, tức là sẽ không chọn phải j}
              c[j] := True;  {Bỏ đánh dấu: j lại là tự do, bởi sắp tới sẽ thử một cách chọn khác của xi}
              end;
      end;
end;

begin
  Assign(Input, 'ARRANGES.INP'); Reset(Input);
  Assign(Output, 'ARRANGES.OUT'); Rewrite(Output);
  ReadLn(n, k);
  FillChar(c, SizeOf(c), True);   {Tất cả các số đều chưa bị chọn}
  Try(1);                         {Thử các cách chọn giá trị của x1}
  Close(Input); Close(Output);
end.
```

Nhận xét: khi k = n thì đây là chương trình liệt kê hoán vị

IV. BÀI TOÁN PHÂN TÍCH SỐ

Bài toán

Cho một số nguyên dương n ≤ 30, hãy tìm tất cả các cách phân tích số n thành tổng của các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là 1 cách.

Cách làm:

1. Ta sẽ lưu nghiệm trong mảng x, ngoài ra có một mảng t. Mảng t xây dựng như sau: t_i sẽ là tổng các phần tử trong mảng x từ x_1 đến x_i: $t_i := x_1 + x_2 + ... + x_i$.
2. Khi liệt kê các dãy x có tổng các phần tử đúng bằng n, để tránh sự trùng lặp ta đưa thêm ràng buộc $x_{i-1} \le x_i$.
3. Vì số phần tử thực sự của mảng x là không cố định nên thủ tục PrintResult dùng để in ra 1 cách phân tích phải có thêm tham số cho biết sẽ in ra bao nhiêu phần tử.
4. Thủ tục đệ quy Try(i) sẽ thử các giá trị có thể nhận của x_i ($x_i \ge x_{i-1}$)
5. Khi nào thì in kết quả và khi nào thì gọi đệ quy tìm tiếp ?

   Lưu ý rằng t_{i-1} là tổng của tất cả các phần tử từ x_1 đến x_{i-1} do đó
   - Khi $t_i = n$ tức là ($x_i = n - t_{i-1}$) thì in kết quả
   - Khi tìm tiếp, x_{i+1} sẽ phải lớn hơn hoặc bằng x_i. Mặt khác t_{i+1} là tổng của các số từ x_1 tới x_{i+1} không được vượt quá n. Vậy ta có $t_{i+1} \le n \Leftrightarrow t_{i-1} + x_i + x_{i+1} \le n \Leftrightarrow x_i + x_{i+1} \le n - t_{i-1}$ tức là x_i

≤ $(n - t_{i-1})/2$. Ví dụ đơn giản khi n = 10 thì chọn x_1 = 6, 7, 8, 9 là việc làm vô nghĩa vì như vậy cũng không ra nghiệm mà cũng không chọn tiếp x_2 được nữa.

Một cách dễ hiểu ta gọi đệ quy tìm tiếp khi giá trị x_i được chọn còn cho phép chọn thêm một phần tử khác lớn hơn hoặc bằng nó mà không làm tổng vượt quá n. Còn ta in kết quả chỉ khi x_i mang giá trị đúng bằng số thiếu hụt của tổng i-1 phần tử đầu so với n.

6. Vậy thủ tục Try(i) thử các giá trị cho x_i có thể mô tả như sau: (để tổng quát cho i = 1, ta đặt x_0 = 1 và t_0 = 0).
   - Xét các giá trị của x_i từ x_{i-1} đến $(n - t_{i-1})$ div 2, cập nhật $t_i := t_{i-1} + x_i$ và gọi đệ quy tìm tiếp.
   - Cuối cùng xét giá trị $x_i = n - t_{i-1}$ và in kết quả từ x_1 đến x_i.

Input: file văn bản ANALYSE.INP chứa số nguyên dương n ≤ 30

Output: file văn bản ANALYSE.OUT ghi các cách phân tích số n.

ANALYSE.INP	ANALYSE.OUT
6	6 = 1+1+1+1+1+1 6 = 1+1+1+1+2 6 = 1+1+1+3 6 = 1+1+2+2 6 = 1+1+4 6 = 1+2+3 6 = 1+5 6 = 2+2+2 6 = 2+4 6 = 3+3 6 = 6

*** Thuật toán quay lui liệt kê các cách phân tích số**

```
program Analyses;
const
  max = 30;
var
  n: Integer;
  x: array[0..max] of Integer;
  t: array[0..max] of Integer;

procedure Init;     {Khởitạo}
begin
  ReadLn(n);
  x[0] := 1;
  t[0] := 0;
end;

procedure PrintResult(k: Integer);
var
  i: Integer;
begin
  Write(n,' = ');
  for i := 1 to k - 1 do Write(x[i], '+');
  WriteLn(x[k]);
end;

procedure Try(i: Integer);
var
  j: Integer;
begin
  for j := x[i - 1] to (n - T[i - 1]) div 2 do     {Trường hợp còn chọn tiếp x[i+1]}
    begin
      x[i] := j;
```

```
      t[i] := t[i - 1] + j;
      Try(i + 1);
    end;
  x[i] := n - T[i - 1];        {Nếu xi là phần tử cuối thì nó bắt buộc phải là ... và in kết quả}
  PrintResult(i);
end;

begin
  Assign(Input, 'ANALYSE.INP'); Reset(Input);
  Assign(Output, 'ANALYSE.OUT'); Rewrite(Output);
  Init;
  Try(1);
  Close(Input);
  Close(Output);
end.
```

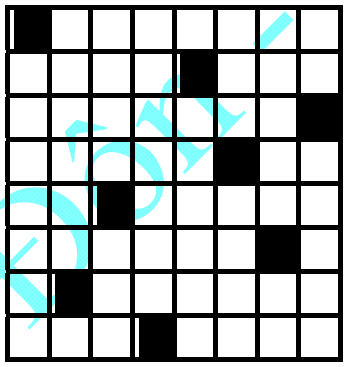
Bây giờ ta xét tiếp một ví dụ kinh điển của thuật toán quay lui:

V. BÀI TOÁN XẾP HẬU

Bài toán

Xét bàn cờ tổng quát kích thước nxn. Một quân hậu trên bàn cờ có thể ăn được các quân khác nằm tại các ô cùng hàng, cùng cột hoặc cùng đường chéo. Hãy tìm các xếp n quân hậu trên bàn cờ sao cho không quân nào ăn quân nào.

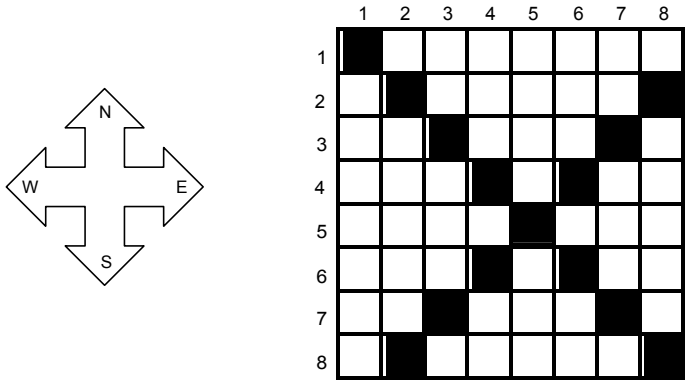
Ví dụ một cách xếp với n = 8:



Hình 3: Xếp 8 quân hậu trên bàn cờ 8x8

Phân tích

- Rõ ràng 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 hậu 1, quân hậu ở hàng 2 là quân hậu 2... quân hậu ở hàng n là quân hậu n. Vậy một nghiệm của bài toán sẽ được biết khi ta tìm ra được **vị trí cột của những quân hậu**.
- Nếu ta định hướng Đông (Phải), Tây (Trái), Nam (Dưới), Bắc (Trên) thì ta nhận thấy rằng:
  - ♦ Một đường chéo theo hướng Đông Bắc - Tây Nam (ĐB-TN) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng + Cột = C (Const). Với mỗi đường chéo ĐB-TN ta có 1 hằng số C và với một hằng số C: $2 \le C \le 2n$ 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
  - ♦ Một đường chéo theo hướng Đông Nam - Tây Bắc (ĐN-TB) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng - Cột = C (Const). Với mỗi đường chéo ĐN-TB ta có 1 hằng số C và với một hằng số C: $1 - n \le C \le n - 1$ 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.



Hình 4: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo ĐN-TB mang chỉ số 0, ô chung (5, 5)

Cài đặt:

1. Ta có 3 mảng logic để đánh dấu:

- Mảng a[1..n]. a_i = TRUE nếu như cột i còn tự do, a_i = FALSE nếu như cột i đã bị một quân hậu khống chế
- Mảng b[2..2n]. b_i = TRUE nếu như đường chéo ĐB-TN thứ i còn tự do, b_i = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.
- Mảng c[1 - n..n - 1]. c_i = TRUE nếu như đường chéo ĐN-TB thứ i còn tự do, c_i = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.
- Ban đầu cả 3 mảng đánh dấu đều mang giá trị TRUE. (Các cột và đường chéo đều tự do)

2. Thuật toán quay lui: 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 đến quân hậu n cho ta 1 nghiệm

3. 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. Điều này có thể kiểm tra ($a_j = b_{i+j} = c_{i-j}$ = TRUE)

4. Khi thử đặt được quân hậu thứ i vào cột j, nếu đó là quân hậu cuối cùng (i = n) thì ta có một nghiệm. Nếu không:

- **Trước khi gọi** đệ quy tìm cách đặt quân hậu thứ i + 1, ta đánh dấu cột và 2 đường chéo bị quân hậu vừa đặt khống chế ($a_j = b_{i+j} = c_{i-j}$:= FALSE) để các lần gọi đệ quy tiếp sau chọn cách đặt các quân hậu kế tiếp sẽ không chọn vào những ô nằm trên cột j và những đường chéo này nữa.
- **Sau khi gọi** đệ quy tìm cách đặt quân hậu thứ i + 1, có nghĩa là sắp tới ta lại thử một cách đặt khác cho quân hậu thứ i, ta bỏ đánh dấu cột và 2 đường chéo bị quân hậu vừa thử đặt khống chế ($a_j = b_{i+j} = c_{i-j}$:= TRUE) tức là cột và 2 đường chéo đó lại thành tự do, bởi khi đã đặt quân hậu i sang vị trí khác rồi thì cột và 2 đường chéo đó hoàn toàn có thể gán cho một quân hậu khác

Hãy xem lại trong các chương trình liệt kê chỉnh hợp không lặp và hoán vị về kỹ thuật đánh dấu. Ở đây chỉ khác với liệt kê hoán vị là: liệt kê hoán vị chỉ cần một mảng đánh dấu xem giá trị có tự do không, còn bài toán xếp hậu thì cần phải đánh dấu cả 3 thành phần: Cột, đường chéo ĐB-TN, đường chéo ĐN- TB. Trường hợp đơn giản hơn: Yêu cầu liệt kê các cách đặt n quân xe lên bàn cờ nxn sao cho không quân nào ăn quân nào chính là bài toán liệt kê hoán vị

Input: file văn bản QUEENS.INP chứa số nguyên dương n ≤ 12

Output: file văn bản QUEENS.OUT, mỗi dòng ghi một cách đặt n quân hậu

QUEENS.INP	QUEENS.OUT
5	(1, 1); (2, 3); (3, 5); (4, 2); (5, 4); (1, 1); (2, 4); (3, 2); (4, 5); (5, 3); (1, 2); (2, 4); (3, 1); (4, 3); (5, 5); (1, 2); (2, 5); (3, 3); (4, 1); (5, 4); (1, 3); (2, 1); (3, 4); (4, 2); (5, 5); (1, 3); (2, 5); (3, 2); (4, 4); (5, 1); (1, 4); (2, 1); (3, 3); (4, 5); (5, 2); (1, 4); (2, 2); (3, 5); (4, 3); (5, 1); (1, 5); (2, 2); (3, 4); (4, 1); (5, 3); (1, 5); (2, 3); (3, 1); (4, 4); (5, 2);

*** Thuật toán quay lui giải bài toán xếp hậu**

```
program n_Queens;
const
  max = 12;
var
  n: Integer;
  x: array[1..max] of Integer;
  a: array[1..max] of Boolean;
  b: array[2..2 * max] of Boolean;
  c: array[1 - max..max - 1] of Boolean;

procedure Init;
begin
  ReadLn(n);
FillChar(a, SizeOf(a), True);  {Mọi cột đều tự do}
FillChar(b, SizeOf(b), True);   {Mọi đường chéo Đông Bắc - Tây Nam đều tự do}
  FillChar(c, SizeOf(c), True);, {Mọi đường chéo Đông Nam - Tây Bắc đều tự do}
end;

procedure PrintResult;
var
  i: Integer;
begin
  for i := 1 to n do Write('(', i, ', ', x[i], '); ');
  WriteLn;
end;

procedure Try(i: Integer); {Thử các cách đặt quân hậu thứ i vào hàng i}
var
  j: Integer;
begin
  for j := 1 to n do
    if a[j] and b[i + j] and c[i - j] then {Chỉ xét những cột j mà ô (i, j) chưa bị khống chế}
      begin
        x[i] := j;              {Thử đặt quân hậu i vào cột j}
        if i = n then PrintResult
        else
          begin
            a[j] := False; b[i + j] := False; c[i - j] := False;   {Đánh dấu}
            Try(i + 1);       {Tìm các cách đặt quân hậu thứ i + 1}
            a[j] := True; b[i + j] := True; c[i - j] := True;      {Bỏ đánh dấu}
          end;
      end;
end;

begin
  Assign(Input, 'QUEENS.INP'); Reset(Input);
  Assign(Output, 'QUEENS.OUT'); Rewrite(Output);
  Init;
```

```
    Try(1);
    Close(Input); Close(Output);
End.
```

Tên gọi thuật toán quay lui, đứng trên phương diện cài đặt có thể nên gọi là kỹ thuật vét cạn bằng quay lui thì chính xác hơn, tuy nhiên đứng trên phương diện bài toán, nếu như ta coi công việc giải bài toán bằng cách xét tất cả các khả năng cũng là 1 cách giải thì tên gọi Thuật toán quay lui cũng không có gì trái logic. Xét hoạt động của chương trình trên cây tìm kiếm quay lui ta thấy tại bước thử chọn x_i nó sẽ gọi đệ quy để tìm tiếp x_{i+1} có nghĩa là quá trình sẽ duyệt tiến sâu xuống phía dưới đến tận nút lá, sau khi đã duyệt hết các nhánh, tiến trình lùi lại thử áp đặt một giá trị khác cho x_i, đó chính là nguồn gốc của tên gọi "thuật toán quay lui"

Bài tập:

1. Một số chương trình trên xử lý không tốt trong trường hợp tầm thường (n = 0 hoặc k = 0), hãy khắc phục các lỗi đó
2. Viết chương trình liệt kê các chỉnh hợp lặp chập k của n phần tử
3. Cho hai số nguyên dương l, n. Hãy liệt kê các xâu nhị phân độ dài n có tính chất, bất kỳ hai xâu con nào độ dài l liền nhau đều khác nhau.
4. Với n = 5, k = 3, vẽ cây tìm kiếm quay lui của chương trình liệt kê tổ hợp chập k của tập {1, 2, ..., n}
5. Liệt kê tất cả các tập con của tập S gồm n số nguyên $\{S_1, S_2, ..., S_n\}$ nhập vào từ bàn phím
6. Tương tự như bài 5 nhưng chỉ liệt kê các tập con có max - min ≤ T (T cho trước).
7. Một dãy $(x_1, x_2, ..., x_n)$ gọi là một hoán vị hoàn toàn của tập {1, 2,..., n} nếu nó là một hoán vị và thoả mãn $x_i \neq i$ với $\forall i$: $1 \leq i \leq n$. Hãy viết chương trình liệt kê tất cả các hoán vị hoàn toàn của tập trên (n vào từ bàn phím).
8. Sửa lại thủ tục in kết quả (PrintResult) trong bài xếp hậu để có thể vẽ hình bàn cờ và các cách đặt hậu ra màn hình.
9. Bài toán mã đi tuần: Cho bàn cờ tổng quát kích thước nxn và một quân Mã, hãy chỉ ra một hành trình của quân Mã xuất phát từ ô đang đứng đi qua tất cả các ô còn lại của bàn cờ, mỗi ô đúng 1 lần.
10. Chuyển tất cả các bài tập trong bài trước đang viết bằng sinh tuần tự sang quay lui.
11. Xét sơ đồ giao thông gồm n nút giao thông đánh số từ 1 tới n và m đoạn đường nối chúng, mỗi đoạn đường nối 2 nút giao thông. Hãy nhập dữ liệu về mạng lưới giao thông đó, nhập số hiệu hai nút giao thông s và d. Hãy in ra tất cả các cách đi từ s tới d mà mỗi cách đi không được qua nút giao thông nào quá một lần.