BÀI TOÁN LIỆT KÊ

Lý thuyết tổ hợp

Nội dung

- Bài toán liệt kê
- Thuật toán và độ phức tạp tính toán
- Giải quyết bài toán liệt kê bằng phương pháp sinh
- Giải quyết bài toán liệt kê bằng phương pháp quay lui.

Bài toán liệt kê

Giới thiệu bài toán

- Bài toán đưa ra danh sách tất cả các cấu hình tổ hợp có thể có được gọi là bài toán liệt kê tổ hợp.
  - Bài toán đếm: tìm kiếm một công thức cho lời giải
  - Bài toán liệt kê: xác định một thuật toán => xây dựng được lần lượt tất cả các cấu hình cần quan tâm.
- Thuật toán liệt kê phải đảm bảo 2 nguyên tắc:
  - Không được lặp lại bất kỳ một cấu hình nào.
  - Không được bỏ sót bất kỳ một cấu hình nào.

Ví dụ 1

- Cho tập hợp các số a_1, a_2, .., a_n và số M. Hãy tìm tất cả các tập con *k* phần tử của dãy số $\{a_n\}$ sao cho tổng số các phần tử trong tập con đó đúng bằng M.
- **Lời giải**:
  - Số các tập con *k* phần tử của tập gồm *n* phần tử là C(n,k).
  - Như vậy, cần phải duyệt trong số C(n,k) tập k phần tử để lấy ra những tập có tổng các phần tử đúng bằng M.
  - Vì không thể xác định được có bao nhiêu tập *k* phần tử từ tập *n* phần tử có tổng các phần tử đúng bằng M => Chỉ còn cách *liệt kê các cấu hình* thoả mãn điều kiện đã cho.

Ví dụ 2

- Một nữ thương nhân đi bán hàng tại tám thành phố. Chị ta có thể bắt đầu hành trình của mình tại một thành phố nào đó nhưng phải qua 7 thành phố kia theo bất kỳ thứ tự nào mà chị ta muốn. Hãy chỉ ra lộ trình ngắn nhất mà chị ta có thể đi.
- **Lời giải**:
  - Vì thành phố xuất phát đã được xác định => thương nhân có thể chọn tuỳ ý 7 thành phố còn lại để hành trình.
  - Như vậy, tất cả số hành trình của thương nhân có thể đi qua là 7! = 5040 cách.
  - Tuy nhiên trong 5040 cách chúng ta phải duyệt toàn bộ để chỉ ra một *hành trình ngắn nhất*.

Thuật toán và độ phức tạp tính toán

Thuật toán

- Dãy hữu hạn các thao tác sơ cấp

$$F = F_1 F_2 .. F_n(\text{Input}) \rightarrow \text{Output}$$

được gọi là một thuật toán trên tập thông tin vào *Input* để có được kết qua ra *Output*.

- Dãy các thao tác sơ cấp được hiểu là các phép toán số học, các phép toán logic, các phép toán so sánh.

Các tính chất của thuật toán

- Tính đơn định
  - Các thao tác sơ cấp phải rõ ràng, không gây nên sự lộn xộn, nhập nhằng, đa nghĩa.
  - Thực hiện đúng các bước của thuật toán trên tập dữ liệu vào, chỉ cho duy nhất một kết quả ra.
- Tính dừng
  - Thuật toán không được rơi vào quá trình vô hạn. Phải dừng lại và cho kết quả sau một số hữu hạn các bước.
- Tính đúng
  - Sau khi thực hiện tất cả các bước của thuật toán theo đúng qui trình đã định, phải nhận được kết quả mong muốn với mọi bộ dữ liệu đầu vào.

Các tính chất của thuật toán (tt)

- Tính phổ dụng
  - Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong lớp các bài toán cùng loại.
  - Có thể làm việc trên nhiều loại dữ liệu khác nhau.
- Tính khả thi
  - Thuật toán phải dễ hiểu, dễ cài đặt.
  - Thực hiện được trên máy tính với thời gian cho phép.

Phương pháp biểu diễn thuật toán

- Ngôn ngữ tự nhiên
- Ngôn ngữ hình thức
  - Giao tiếp trung gian giữa con người và hệ thống máy tính.
  - Ví dụ: ngôn ngữ sơ đồ khối, ngôn ngữ tựa tự nhiên, ngôn ngữ đặc tả.
  - Rất gần với ngôn ngữ tự nhiên và ngôn ngữ máy tính.
- Ngôn ngữ máy tính
  - Giao tiếp giữa máy tính và máy tính.
  - Có thể sử dụng bất kỳ ngôn ngữ lập trình nào để mô tả thuật toán.

Ví dụ 1. Biểu diễn thuật toán tìm USCLN (a, b) bằng ngôn ngữ tự nhiên

Đầu vào (Input). Hai số tự nhiên a, b.

Đầu ra (Output). Số nguyên u lớn nhất để a và b đều chia hết cho u.

Thuật toán (Euclide Algorithm):

Bước 1. Đưa vào hai số tự nhiên a và b.

Bước 2. Nếu b≠ 0 thì chuyển đến bước 3, nếu b=0 thì thực hiện bước 4.

Bước 3. Đặt r = a mod b; a = b; b = r ; Sau đó quay trở lại bước 2.

Bước 4 (Output). Kết luận u=a là số nguyên cần tìm.

r = a mod b = a%b

Ví dụ 2. Biểu diễn thuật toán tìm USCLN (a, b) bằng ngôn ngữ hình thức

Thuật toán Euclide:

Đầu vào (*Input*): $a \in N$, $b \in N$.

Đầu ra (*Output*): $s = \max \{ u \in N : a \bmod u = 0 \text{ and } b \bmod u = 0 \}$.

Format : s = Euclide (a, b).

Actions :

```
while (b ≠ 0 )  do
        r = a mod b; a = b; b = r;
endwhile;
return(a);
```

Endactions.

Ví dụ 3. Biểu diễn thuật toán tìm USCLN (a, b) bằng ngôn ngữ máy tính (C++)

```
Int    USCLN( int  a, int  b) {
       while ( b != 0 ) {
               r = a % b; a = b; b = r;
             }
             return(a);
       }
```

Độ phức tạp tính toán

- Thời gian thực hiện một giải thuật bằng chương trình máy tính phụ thuộc vào các yếu tố:
  - Kích thước dữ liệu vào.
  - Phần cứng máy tính.

Độ phức tạp thuật toán

- Cho hai hàm *f(x)*, *g(x)* xác định trên tập các số nguyên dương hoặc tập các số thực vào tập các số thực.
- Hàm *f(x)* được gọi là *O(g(x))* nếu tồn tại một hằng số C>0 và n_0 sao cho:

$$|f(x)| \le C.|g(x)| \text{ vớ mọi } x \ge n_0.$$

- Nếu *f(x)* là thời gian thực hiện của một thuật toán thì ta nói giải thuật đó có cấp *g(x)* hay độ phức tạp thuật toán là *O(g(x))*.

Ví dụ 1

- Cho: $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$
  - a_i là các số thực (i =0,1, 2, ..,n).
  - Khi đó: $f(x) = O(x^n)$

Chứng minh. Thực vậy, với mọi x>1:

$$|f(x)| = |a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0|$$

$$\cdots \le |a_n| x^n + |a_{n-1}| \underline{x^{n-1}} + \ldots + |a_1| x + |a_0|$$

phải lấy giá trị lớn nhất $\cdots \le |a_n| x^n + |a_{n-1}| \underline{x^n} + \ldots + |a_1| x^n + |a_0| x^n$

$$\cdots \le x^n (|a_n| + |a_{n-1}| + \ldots + |a_1| + |a_0|)$$

$$\cdots \le C.x^n = O(x^n).$$

$$C = (|a_n| + |a_{n-1}| + \ldots + |a_1| + |a_0|)$$

Ví dụ 2 - Tìm độ phức tạp thuật toán sắp xếp kiểu Bubble-Sort?

sắp xếp từ bé đến lớn

```
Void Bubble-Sort ( int  A[],  int n ) {
        for ( i=1; i<n; i++) {
                for ( j = i+1; j<=n; j++){
                        if (A[i] > A[j]) {
                                t = A[i]; A[i] = A[j]; A[j] = t;
                        }
                }
        }
}
```

Lời giải. Sử dụng trực tiếp nguyên lý cộng ta có:

- Với i =1 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- Với i = 2 ta cần sử dụng n-2 phép so sánh A[i] với A[j];
- . . . . .
- Với i = n-1 ta cần sử dụng 1 phép so sánh A[i] với A[j];

Vì vậy tổng số các phép toán cần thực hiện là:

$$S = (n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2 \le n^2 = O(n^2).$$

Tính chất của độ phức tạp thuật toán

- Với P(n) là một đa thức bậc *k* thì O(P(n)) = O(n^k).
  - Thuật toán có độ phức tạp cấp đa thức là **O(n^k)**.
- Với a, b là hai cơ số tùy ý và f(n) là một hàm xác định dương thì $\boxed{\log_a f(n)=\log_a b.\log_b(f(n)).}$
  - Độ phức tạp thuật toán cấp logarit được ký hiệu là **O(log(f(n))** mà không cần quan tâm đến cơ số.
- Nếu độ phức tạp thuật toán là hằng số,
  - Nghĩa là thời gian tính toán không phụ thuộc vào độ dài dữ liệu
  - Ký hiệu là **O(1)**.

Tính chất của độ phức tạp thuật toán (tt)

- Một giải thuật có cấp 2^n, n!, n^n được gọi là giải thuật *hàm mũ*.
  - Những giải thuật này thường có tốc độ rất chậm.
- Độ phức tạp tính toán của một đoạn chương trình *P* chính bằng *số lần thực hiện một phép toán tích cực*.
  - Phép toán tích cực trong một đoạn chương trình là phép toán mà *số lần thực hiện nó không ít hơn các phép toán khác*.

Các dạng hàm đánh giá độ phức tạp thuật toán

tăng dần về
mặt thời gian

Dạng đánh giá	Tên gọi
O(1)	Hằng số
O(lg lg n)	Log log
O(lg n)	Logarithm
O(n)	Tuyến tính
O(n lg n)	n log
$O(n^2)$	Bậc hai
$O(n^3)$	Bậc 3
$O(n^m)$	Đa thức
$O(m^n)$ $m \geq 2$	Hàm mũ
O(n!)	Giai thừa

n là độ dài input

Qui tắc xác định độ phức tạp thuật toán

- **Qui tắc tổng**: Nếu $f_1(x)$ có độ phức tạp là $O(g_1(x))$ và $f_2(x)$ có độ phức tạp là $O(g_2(x))$ thì độ phức tạp của $f_1(x) + f_2(x)$ là $O(Max(g_1(x), g_2(x)))$.

Chứng minh.

- Vì $f_1(x)$ có độ phức tạp là $O(g_1(x))$ nên tồn tại hằng số C_1 và k_1 sao cho $|f_1(x)| \le C_1|g_1(x)|$ với mọi $x \ge k_1$;
- Vì $f_2(x)$ có độ phức tạp là $O(g_2(x))$ nên tồn tại hằng số C_2 và k_2 sao cho $|f_2(x)| \le |C_2 g_2(x)|$ với mọi $x \ge k_2$;
- Ta lại có :

$$|f_1(x)+ f_2(x)| \le |f_1(x)| + |f_2(x)|$$
$$\le C_1|g_1(x)| + C_2|g_2(x)|$$
$$\le C|g(x)| \text{ với mọi } x > k;$$

Trong đó, $C = C_1 + C_2$; $g(x) = \max(g_1(x), g_2(x))$; $k = \max (k_1, k_2)$.

- **Tổng quát**: Nếu độ phức tạp của $f_1(x), f_2(x),.., f_m(x)$ lần lượt là $O(g_1(x)), O(g_2(x)),.., O(g_m(x))$ thì độ phức tạp của $f_1(x) + f_2(x) + ..+f_m(x)$ là $O(max(g_1(x), g_2(x),..,g_m(x))$.

Qui tắc xác định độ phức tạp thuật toán (tt)

- **Qui tắc nhân**: Nếu f(x) có độ phức tạp là O(g(x)) thì độ phức tạp của $f^n(x)$ là $O(g^n(x))$, trong đó:

  $f^n(x) = f(x).f(x)\ldots.f(x)$. //n lần f(x)

  $g^n(x) = g(x).g(x)\ldots g(x)$. //n lần g(x)

- Nói cách khác, đoạn chương trình *P* có thời gian thực hiện T(n)= O(f(n)).
  - Khi đó, nếu thực hiện *k(n)* lần đoạn chương trình *P* với *k(n) là O(g(n))* thì độ phức tạp tính toán là O(f(n).g(n)).

Chứng minh. Thật vậy theo giả thiết f(x) là O(g(x)) nên tồn tại hằng số C và k sao cho với mọi x>k thì $|f(x)| \le C.|g(x)|$. Ta có:

$$
\begin{aligned}
\left|f^n(x)\right| &= \left|f(x).f(x)\ldots f(x)\right| \qquad \textit{// nhân n lần f(x)} \\
&\le \left|C.g(x).C.g(x)\ldots C.g(x)\right| \\
&\le C^n \left|g^n(x)\right| = O\left(g^n(x)\right)
\end{aligned}
$$

Phương pháp sinh

- Để giải các bài toán liệt kê, cần thỏa:
  1. Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê.
     - Từ đó có thể xác định được cấu hình tổ hợp đầu tiên và cuối cùng trong thứ tự đã được xác định.
  2. Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có để đưa ra cấu hình kế tiếp sau nó.

Phương pháp sinh kế tiếp

Mã giả

```
procedure Generation ( ){
        <Xây dựng cấu hình ban đầu>;
        stop =false
        while (! stop) { kt khi nào thuật toán sẽ dừng
                <Đưa ra cấu hình đang có>;  liệt kê
                <Sinh ra cấu hình kế tiếp>;  quy luật
        }
}
```

Ví dụ 1. Liệt kê tất cả các dãy nhị phân độ dài *n*

Lời giải. Viết dãy nhị phân dưới dạng $b_1b_2..b_n$, trong đó $b_i \in \{0, 1\}$. Xem mỗi dãy nhị phân $b=b_1b_2..b_n$ là biểu diễn nhị phân của một số nguyên p(b). Khi đó thứ tự hiển nhiên nhất có thể xác định trên tập các dãy nhị phân là thứ tự từ điển được xác định như sau:

Ta nói dãy nhị phân $b = b_1b_2..b_n$ đi trước dãy nhị phân $b' = b'_1b'_2..b'_n$ theo thứ tự từ điển và ký hiệu b<b'nếu p(b) <p(b').

Ví dụ với n=4, các xâu nhị phân độ dài 4 được liệt kê theo thứ tự từ điển là:

b	p(b)	b	p(b)
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Ví dụ 1. Liệt kê tất cả các dãy nhị phân độ dài *n* (tt)

- **Nhận xét**: Dãy đầu tiên là 0000, dãy cuối cùng là 1111.
  - Nếu xâu nhị phân chứa toàn bít 1 thì quá trình liệt kê kết thúc,
  - Trái lại, dãy kế tiếp sẽ nhận được bằng cách cộng thêm 1 vào dãy hiện tại.
- **Qui tắc sinh kế tiếp**:
  - Tìm *i* đầu tiên từ tay phải sang trái (i=n, n-1, . .,1) thoả mãn b_i =0.
  - Gán lại b_i =1 và b_j=0 với tất cả j>i. Dãy thu được là dãy cần tìm.
- Chẳng hạn với xâu nhị phân độ dài 10: 1100111011.
  - i = 8, ta đặt lại b_8 =1, b_9,b_{10} =0
  - Xâu nhị phân kế tiếp: 1100111100.

1 ← *n*

b_8

Ví dụ 1. Thuật toán sinh kế tiếp

```
void Next_Bit_String( void ){
        int i = n;//Xuất phát tại i=n
        while (i> 0 && b_i !=0 ) { //Nếu b_i=1 thì gán thành 0
                b_i = 0; i = i-1;
        }
        if ( i > 0 )  b_i = 1; //Nếu i>0 thì chưa là cấu hình cuối cùng
        else OK = False; ///Nếu i=0 thì là cấu hình cuối cùng  => Dừng.
}
```

Ví dụ 1. Chương trình liệt kê các xâu nhị phân có độ dài *n*

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
using namespace std;
int n, X[MAX], OK=TRUE, dem=0;
void Init (void ){
        cout<<"\n Nhap n=";cin>>n;
        for (int i=1; i<=n; i++)    X[i] =0;
}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=n; i++)
                cout<<X[i]<<"   ";
}
void Next_Bit_String(void) {
         int i= n;
        while (i>0 && X[i]!=0){   X[i] = 0; i --;   }
        if (i > 0 ) X[i] = 1;
        else OK = FALSE;
}
int main() {
        Init(); //Nhap n = 4
        while (OK ){
                Result();
                Next_Bit_String();
        }
        system("PAUSE");   return 0;
}
```

Ví dụ 1. Kết quả thực hiện chương trình

```
Nhap n=4
Ket qua buoc 1:0   0   0   0
Ket qua buoc 2:0   0   0   1
Ket qua buoc 3:0   0   1   0
Ket qua buoc 4:0   0   1   1
Ket qua buoc 5:0   1   0   0
Ket qua buoc 6:0   1   0   1
Ket qua buoc 7:0   1   1   0
Ket qua buoc 8:0   1   1   1
Ket qua buoc 9:1   0   0   0
Ket qua buoc 10:1   0   0   1
Ket qua buoc 11:1   0   1   0
Ket qua buoc 12:1   0   1   1
Ket qua buoc 13:1   1   0   0
Ket qua buoc 14:1   1   0   1
Ket qua buoc 15:1   1   1   0
Ket qua buoc 16:1   1   1   1
```

Ví dụ 2. Liệt kê tập con *k* phần tử của tập *n* phần tử

C_n^k

- Cho X = { 1, 2, . ., n }. Hãy liệt kê tất cả các tập con *k* phần tử của X (k≤n)

Lời giải: Mỗi tập con của tập hợp X có thể biểu diễn bằng bộ có thứ tự gồm k thành phần $a = (a_1 a_2 .. a_k)$ thoả mãn $1 \le a_1 \le a_2 \le . . \le a_k \le n$. Trên tập các tập con k phần tử của X có thể xác định nhiều thứ tự khác nhau. Thứ tự dễ nhìn thấy nhất là thứ tự từ điển được định nghĩa như sau:

Ta nói tập con $a = a_1 a_2 . . . a_k$ đi trước tập con $a' = a_1' a_2' . . . a_k'$ trong thứ tự từ điển và ký hiệu là $a < a'$, nếu tìm được chỉ số j ($1 \le j \le k$) sao cho

$a_1 = a_1', a_2 = a_2', \ . . ., a_{j-1} = a'_{j-1}, a_j < a'_j$.

Ví dụ 2. Liệt kê tập con k phần tử của tập n phần tử

- Chẳng hạn X = { 1, 2, 3, 4, 5 }, k = 3.
  - Các tập con 3 phần tử của X được liệt kê theo thứ tự từ điển như sau:

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

Ví dụ 2... (tt)

- **Nhận xét**:
  - Tập con đầu tiên trong thứ tự từ điển là (1, 2, . ., k)
  - Tập con cuối cùng là (n-k+1, n-k+2, . ., n).
- $a = (a_1, a_2, . ., a_k)$ là tập con hiện tại và chưa phải là cuối cùng, => Qui tắc sinh kế tiếp:
  - Tìm từ bên tay phải dãy $a_1, a_2, . ., a_k$ phần tử $a_i \neq n - k + i$
  - Thay a_i bởi $a_i + 1$,
  - Thay a_j bởi $a_i + j - i$, với j:= i+1, i + 2, . . ., k
- Chẳng hạn với n = 6, k =4.
  - Giả sử ta đang có tập con (1, 2, 5, 6), => Quy tắc sinh:
    - Duyệt từ bên phải, được i =2, thay a_2 bởi $a_2 + 1 = 2 + 1 = 3$.
    - Duyệt j từ i + 1 = 3 cho đến k, thay thế $a_3 = a_2 + 3 - 2 = 3 + 3 - 2 = \mathbf{4}$, $a_4 = a_2 + 4 - 2 = 3 + 4 - 2 = \mathbf{5}$ => tập con kế tiếp theo thứ tự từ điển là **(1, 3, 4, 5)**. → 1, 3, 4, 6 → 1, 3, 5, 6

Ví dụ 2. Thuật toán liệt kê tập con kế tiếp k phần tử của tập n phần tử

```
void Next_Combination(void){
    i = k; //Xuất phát từ vị trí thứ k
    while ( i> 0 && a_i == n-k+i) //Xác định i để a_i ≠ n-k+i
        i = i -1;
    if (i>0) { //Nếu i>0 thì chưa phải là tổ hợp cuối cùng
        a_i = a_i + 1;
        for ( j = i+1; j <=k; j++)
            a_j = a_i + j - i;
    }
    else OK =False; ////Nếu i=0 thì là tổ hợp cuối cùng
```

Ví dụ 2. Chương trình liệt kê tổ hợp chập k của 1, 2, .., n

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,k,X[MAX],OK=TRUE,dem=0;
void Init (void ){
         cout<<"\n Nhap n=";cin>>n;
        cout<<"\n Nhap k=";cin>>k;
        for (int i=1; i<=k; i++)
                X[i] =i;
}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=k; i++)
                cout<<X[i]<<"   ";
}
```

```
void Next_Combination(void) {
        int i= k;
        while (i>0 && X[i]==n-k+i)   i--;
        if (i > 0 ) {
                X[i] = X[i] +1;
                for (int j = i+1; j<=k; j++)
                        X[j] = X[i] + j - i;
        }
        else OK = FALSE;
}
int main() {
        Init(); //Nhap n = 5, k = 3
        while (OK ){
                Result();
                Next_Combination();
        }
        system("PAUSE");  return 0;
}
```

Ví dụ 2. Kết quả thực hiện chương trình với n = 5, k= 3

```
Nhap n=5
Nhap k=3
Ket qua buoc 1:1   2   3
Ket qua buoc 2:1   2   4
Ket qua buoc 3:1   2   5
Ket qua buoc 4:1   3   4
Ket qua buoc 5:1   3   5
Ket qua buoc 6:1   4   5
Ket qua buoc 7:2   3   4
Ket qua buoc 8:2   3   5
Ket qua buoc 9:2   4   5
Ket qua buoc 10:3   4   5
```

Ví dụ 3. Liệt kê các hoán vị của tập ***n*** phần tử

- Cho X = { 1, 2, .., n }. Hãy liệt kê các hoán vị từ *n* phần tử của X.

Lời giải : Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự n thành phần

$a = (a_1, a_2, .., a_n)$ thoả mãn $a_i \in X$, $i = 1, 2, .., n$, $a_p \neq a_q$, $p \neq q$.

Trên tập các hoán vị từ n phần tử của X có thể xác định nhiều thứ tự khác nhau. Tuy nhiên, thứ tự dễ thấy nhất là thứ tự từ điển được định nghĩa như sau:

Ta nói hoán vị $a = a_1a_2. . . a_n$ đi trước hoán vị $a' = a_1'a_2'. . .a_n'$ trong thứ tự từ điển và ký hiệu là $a<a'$, nếu tìm được chỉ số k ($1 \le k \le n$) sao cho

$a_1 = a_1'$, $a_2 = a_2'$, . . ., $a_{k-1} = a'_{k-1}$, $a_k < a'_k$.

Ví dụ 3. Liệt kê các hoán vị của tập *n* phần tử

- Chẳng hạn X = {1, 2, 3},
  - Các hoán vị các phần tử của X được liệt kê theo thứ tự từ điển như sau:

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

Ví dụ 3... (tt)

- **Nhận xét**:
  - Hoán vị đầu tiên trong thứ tự từ điển là (1, 2, …, n)
  - Hoán vị cuối cùng là (n, n-1, . . ., 1)
- $a = (a_1, a_2, . . ., a_n)$ là một hoán vị chưa phải là cuối cùng, => Qui tắc sinh kế tiếp:
  - Tìm từ tay phải qua trái hoán vị có chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (j là chỉ số lớn nhất để $a_j < a_{j+1}$);
  - Tìm a_k là số nhỏ nhất còn lớn hơn a_j trong các số ở bên phải a_j;
  - Đổi chỗ a_j với a_k;
  - Lật ngược đoạn từ a_{j+1} đến a_n.
- Chẳng hạn đang có hoán vị (3, 6, 2, 5, 4, 1) => Quy tắc sinh:
  - Duyệt từ j= n-1 từ tay phải sang trái để tìm j đầu tiên thoả $a_j<a_{j+1}$ => j=3 ($a_3=2<a_4=5$).
  - Số nhỏ nhất còn lớn hơn a_3 trong các số bên tay phải a_3 là a_5 ($a_5=4$).
  - Đổi chỗ a_3 cho a_5 => (3, 6, 4, 5, 2, 1)
  - Lật ngược đoạn từ a_4 đến a_6 => (3,6,4,1,2,5).

Ví dụ 3. Thuật toán sinh hoán vị kế tiếp

void Next_Permutation(void){

  j = n-1; //Duyệt từ vị trí j=n-1

  while (j> 0 && $a_j > a_{j+1}$) //Tìm vị trí j để $a_j > a_{j+1}$

    j = j -1;

  if (j>0) { // Nếu j>0 thì hoán vị chưa phải cuối cùng

    k = n; //Xuất phát từ vị trí k=n

    while ($a_j > a_k$) // Tìm k để $a_j < a_k$

      k= k - 1;

    temp =a_j; $a_j = a_k$; a_k = temp;//Đổi chỗ a_j cho a_k

    r = j + 1; s = n;

    while (r < s) {//Lật ngược lại đoạn từ j+1 đến n

      temp = a_r; $a_r = a_s$; a_s = temp;

      r = r +1;   s = s - 1;

    }

  }

  else OK = False;//Nếu là hoán vị cuối cùng thì i=0

}

Ví dụ 3. Chương trình liệt kê hoán vị

```
#include <iostream.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,X[MAX],OK=TRUE,dem=0;
void Init (void ){
     cout<<"\n Nhap n=";cin>>n;
     for (int i=1; i<=n; i++)
         X[i] =i;
}
void Result(void){
     cout<<"\n Ket qua buoc "<<++dem<<":";
     for (int i=1; i<=n; i++)
         cout<<X[i]<<"   ";
}
```

```
void Next_Permutation(void) {
     int j= n-1;
     while (j>0 && X[j]>X[j+1])j--;
     if (j > 0 ) {
         int k =n;
         while(X[j]>X[k]) k--;
         int t = X[j]; X[j]=X[k]; X[k]=t;
         int r = j +1, s =n;
         while (r<=s ) {
               t = X[r]; X[r]=X[s]; X[s] =t;
               r ++; s--;
         }
     }
     else OK = FALSE;

int main()
{
      Init(); //Nhap n = 3
      while (OK ){
            Result();
            Next_Permutation();
      }
      system("PAUSE");
      return 0;
}
```

Ví dụ 3. Kết quả thực hiện chương trình hoán vị với n=3

```
Nhap n=3

Ket qua buoc 1:1    2    3
Ket qua buoc 2:1    3    2
Ket qua buoc 3:2    1    3
Ket qua buoc 4:2    3    1
Ket qua buoc 5:3    1    2
Ket qua buoc 6:3    2    1
```

Thuật toán quay lui (Back track)

- **Nguyên lý**: xây dựng dần các thành phần của cấu hình bằng cách thử tất cả các khả năng.
- **Bài toán**:
  - Cần tìm một cấu hình $x = (x_1, x_2, . . ., x_n)$ mà *i-1* thành phần $x_1, x_2, . . ., x_{i-1}$ đã được xác định,
  - Xác định thành phần thứ *i* của cấu hình bằng cách duyệt tất cả các khả năng có thể có và đánh số các khả năng từ *1. .n*$_i$. Với mỗi khả năng *j*, có thể:
    - Nếu chấp nhận *j* thì xác định x_i theo *j*,
      - nếu i=n thì ta được một cấu hình cần tìm,
      - ngược lại xác định tiếp thành phần x_{i+1}.
    - Nếu không có khả năng nào được chấp nhận:
      - quay lại bước trước đó để xác định lại x_{i-1}.

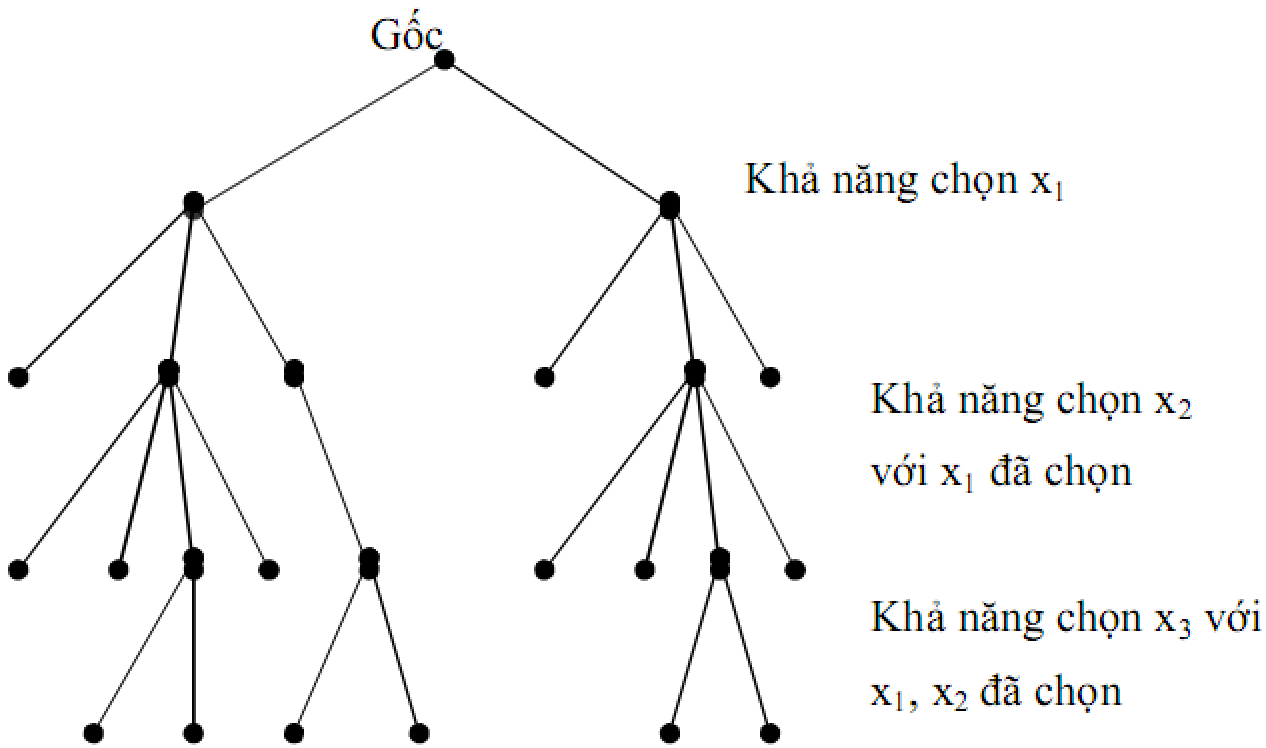
Thuật toán quay lui (tt)

- **Quan trọng**: nhớ lại mỗi bước đã đi qua để tránh sự trùng lặp.
  - Tổ chức theo cơ chế Ngăn xếp (LIFO).
  - Phù hợp với phép gọi **đệ qui.**

```
void Try( int i )  {
        for (j = 1; j < n_i; j ++) {
                if ( <Chấp nhận j >) {
                        <Xác định x_i theo j>
                        if (i==n)
                                <Ghi nhận cấu hình>;
                        else    Try(i+1);
                }
        }
}
```

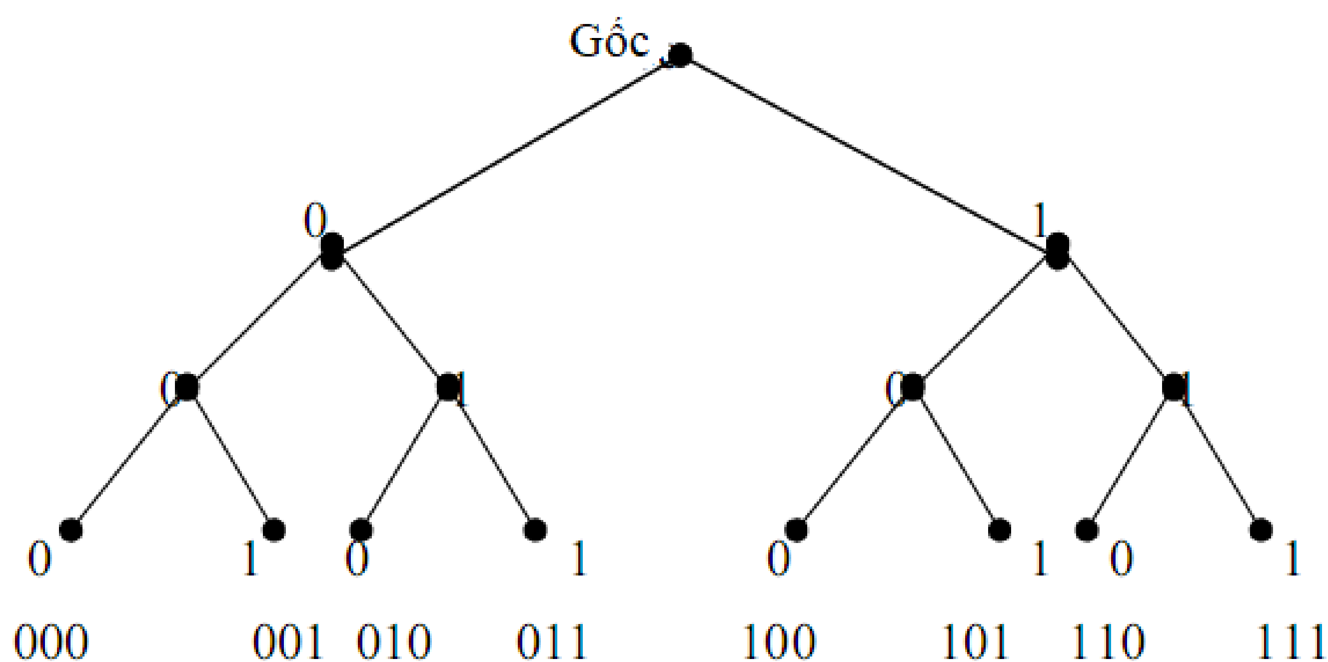
Cây tìm kiếm lời giải

- Cây liệt kê lời giải theo thuật toán quay lui:



Ví dụ 1. Liệt kê các xâu nhị phân độ dài *n*

- Biểu diễn các xâu nhị phân dưới dạng $x_1, x_2, \ldots, x_n$, trong đó $x_i \in \{0, 1\}$.
- Với n =3, cây tìm kiếm lời giải được thể hiện như sau:



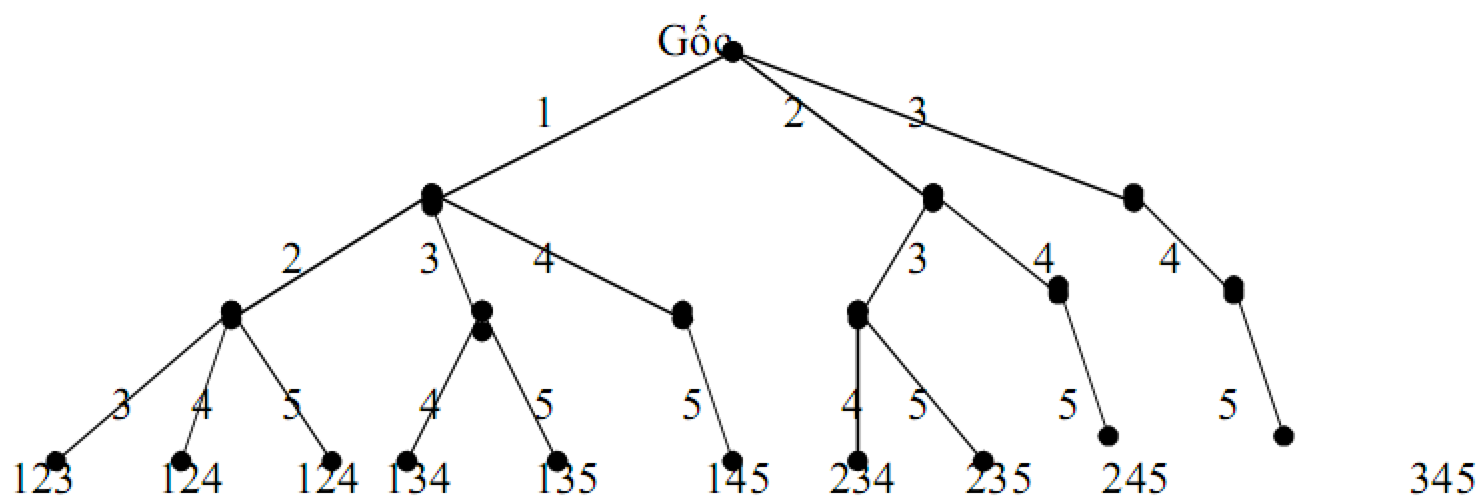
Ví dụ 1. Chương trình liệt kê các xâu nhị phân có độ dài n sử dụng thuật toán quay lui

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n, X[MAX], dem=0;
void Init (void ){
        cout<<"\n Nhap n=";cin>>n;
}
void Result(void){
    cout<<"\n Ket qua buoc "<<++dem<<":";
    for (int i=1; i<=n; i++)
        cout<<X[i]<<"   ";
}
```

```
void Try (int i) {
    for (int j=0; j<=1; j++){
        X[i] = j;
        if (i==n) Result();
        else Try(i+1);
    }
}
int main(){
    Init(); //Nhap n = 3
    Try(1);system("PAUSE");return 0;
}
```

Ví dụ 2. Liệt kê các tập con *k* phần tử của tập ***n*** phần tử dùng Back track

- Biểu diễn tập con k phần tử dưới dạng $x_1, x_2, . ., x_k$, trong đó $1 \le x_1 < x_2 \; . . \le n$.
  => Các giá trị đề cử cho x_i là từ x_{i-1} + 1 cho đến n - k + i.
- Cây tìm kiếm lời giải với n=5, k=3:



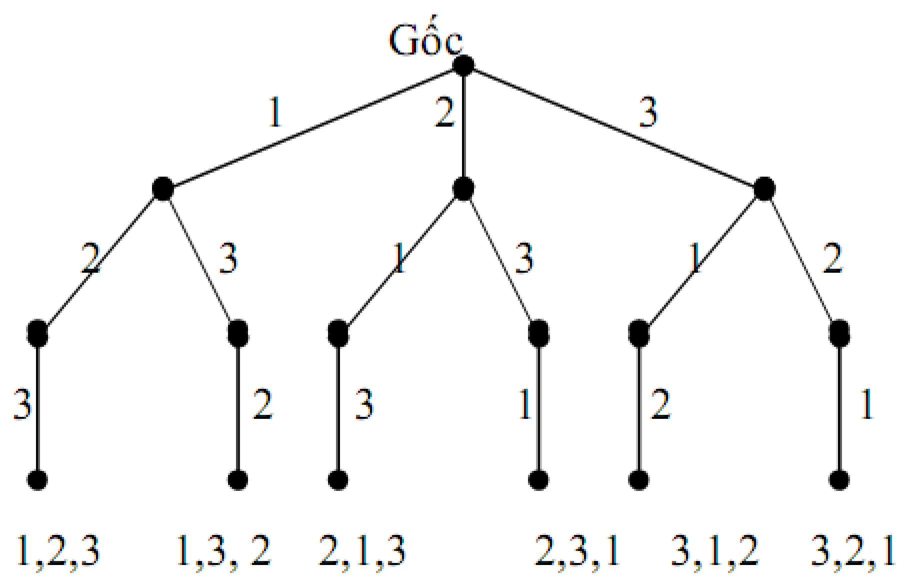
Ví dụ 2. Chương trình liệt kê các tập con *k* phần tử trong tập *n* phần tử dùng Back track C_n^k

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,k,X[MAX],dem=0;
void Init (void ){
        cout<<"\n Nhap n=";cin>>n;
        cout<<"\n Nhap k=";cin>>k;
        X[0] = 0;
}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=k; i++)
                cout<<X[i]<<"   ";
}
void Try(int i ) {
        for (int j = X[i-1]+1; j<= n-k+i; j++){
                X[i] = j;
                if (i==k) Result();
                else Try(i+1);
        }
}
int main(){
    Init(); //Nhap n = 5, k = 3
    Try(1);
    system("PAUSE");
    return 0;
}
```

Ví dụ 3. Liệt kê các hoán vị của tập *n* phần tử dùng Back track

- Biểu diễn hoán vị dưới dạng $\mathbf{x_1, x_2, .., x_n}$, trong đó $\mathbf{x_i}$ nhận giá trị từ 1 đến *n* và $x_i \neq x_j$ với $i \neq j$.
- Các giá trị từ 1 đến *n* lần lượt được đề cử cho $\mathbf{x_i}$, trong đó giá trị *j* được chấp nhận nếu nó chưa được dùng.

  => Lưu ý: với mỗi giá trị *j* xem nó đã được dùng hay chưa.
- Cây tìm kiếm lời giải với n = 3



Ví dụ 3. Chương trình giải quyết bài toán liệt kê các hoán vị của 1, 2, . ., n

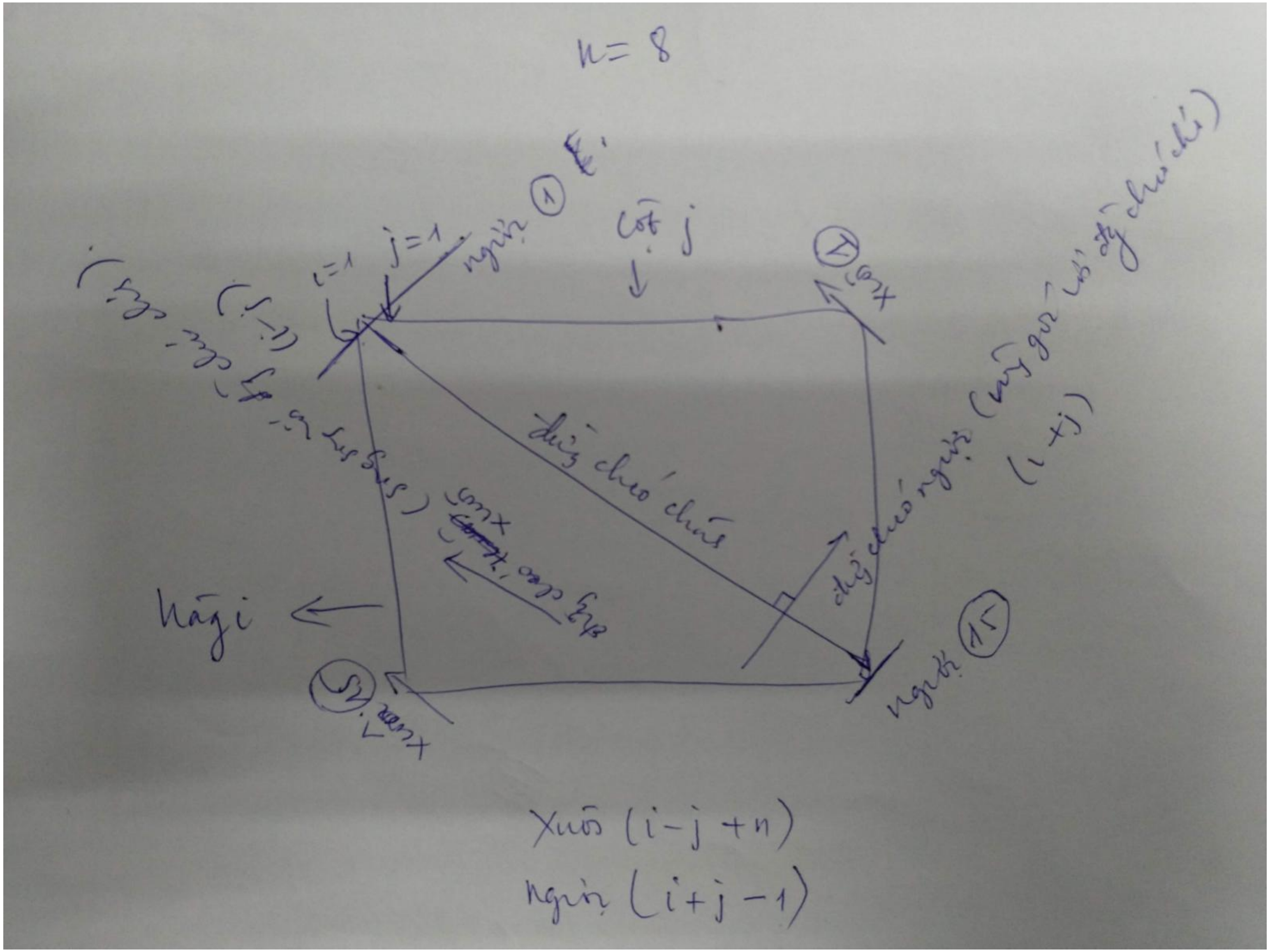
```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,X[MAX],chuaxet[MAX],dem=0;
void Init (void ){
    cout<<"\n Nhap n=";cin>>n;
    for (int i=1; i<=n; i++)   chuaxet[i] = TRUE;
}
void Result(void){
    cout<<"\n Ket qua buoc "<<++dem<<":";
    for (int i=1; i<=n; i++)   cout<<X[i]<<"   ";
}
```

```
void Try(int i){
        for (int j =1; j<=n; j++){
                if (chuaxet[j]) {
                        X[i] = j; chuaxet[j] = FALSE;
                        if (i == n) Result();
                        else Try(i+1);
                        chuaxet[j] = TRUE;  reset
                }
        }
}
int main(){   Init(); //Nhap n = 3
        Try (1);  system("PAUSE");  return 0;
}
```

Ví dụ 4. Bài toán Xếp Hậu

- Liệt kê tất cả các cách xếp n quân hậu trên bàn cờ n x n sao cho chúng không ăn được nhau.
- Giải thích bài toán:
  – Bàn cờ có *n* hàng được đánh số từ 1 đến n, *n* cột được đánh số từ 1 đến n
  – Bàn cờ có 2*n -1 đường chéo *xuôi* được đánh số từ 1 đến 2*n -1
  – 2 *n -1 đường chéo *ngược* được đánh số từ 1 đến 2*n -1.

Tham khảo


n = 8
i=1 j=1
cột j
ngược 1
(i-j)
xuôi 15
ngược 15
xuôi 1
hàng i
đường chéo chính
Xuôi (i-j+n)
Ngược (i+j-1)

Ví dụ 4. Phân tích bài toán

- Vì trên mỗi hàng chỉ xếp được đúng một quân hậu => chỉ cần quan tâm đến quân hậu được xếp ở cột nào.
- => Việc xác định bộ *n* thành phần $x_1, x_2, . ., x_n$, trong đó $x_i = j$ được hiểu là quân hậu (x) tại dòng *i* xếp vào cột thứ *j*.
  - Giá trị của *i* được nhận từ 1 đến *n*;
  - Giá trị của *j* cũng được nhận từ 1 đến *n*, nhưng thoả mãn điều kiện:
    - ô (i,j) chưa bị quân hậu khác chiếu đến theo cột, đường chéo xuôi, đường chéo ngược.

Ví dụ 4. Chương trình Xếp Hậu bằng thuật toán quay lui

```
#include <iostream.h>
#include <stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int X[MAX], XUOI[MAX], NGUOC[MAX], chuaxet[MAX];
int n, dem =0;
void Init (void ) {
        cout<<"\n Nhap n ="; cin>>n;
        for (int i=1; i<=n; i++) chuaxet[i]=TRUE;
        for (int i=1; i<=(2*n-1); i++) {
                XUOI[i] = TRUE; NGUOC[i]=TRUE;
        }
}
void Result(void ) {
        cout<<"\n Phuon an "<<++dem<<":";
        for (int i=1; i<=n; i++)        cout<<X[i]<<"  ";
}
```

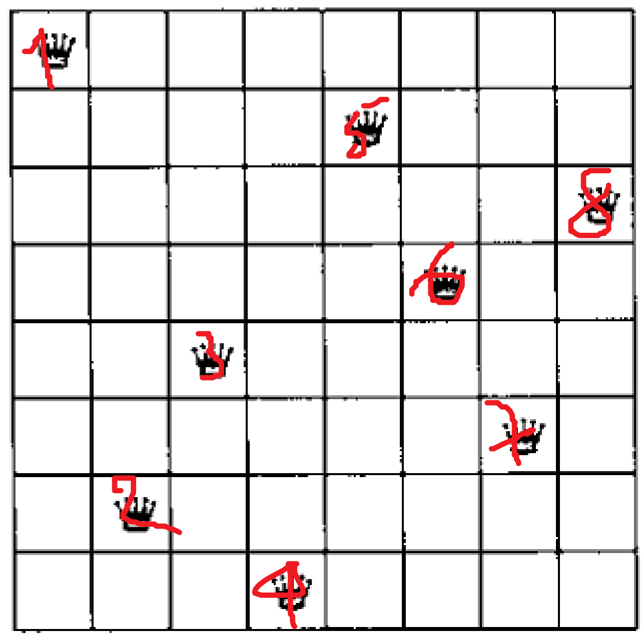
Ví dụ 4. Chương trình Xếp Hậu bằng thuật toán quay lui (tt)

```
void Try(int i){
        for (int j =1; j<=n; j++){
                if (chuaxet[j] && XUOI[i-j+n] && NGUOC[i+j-1]){
                        X[i] = j;chuaxet[j]=FALSE;
                        XUOI[i-j+n]=FALSE;NGUOC[i+j-1]=FALSE;
                        if (i==n ) Result();
                        else Try(i+1);
                        chuaxet[j]=TRUE;XUOI[i-j+n]=TRUE;
                        NGUOC[i+j-1]=TRUE;
                }
        }
}
int main(){
        Init(); Try(1);
        system("PAUSE");
        return 0;
}
```

Ví dụ 4. Số cách xếp hậu ứng với n

n	4	7	8	9	10	11	12	13	14
H_n	2	40	92	352	724	2680	14200	73712	365596

Nghiệm đầu tiên mà chương trình tìm được ứng với n =8 là x =(1, 5, 8, 6, 3, 7, 2, 4).



Bài tập 1

- Liệt kê tất cả các chuỗi nhị phân độ dài 5 không chứa hai số 0 liên tiếp.

Bài tập 2

- Cho *n* là số nguyên dương. Một cách phân chia số *n* là biểu diễn *n* thành tổng các số tự nhiên không lớn hơn *n*. Chẳng hạn 7 = 2 + 3 + 2.
- Xây dựng thuật toán và cài đặt để liệt kê tất cả các cách chia có thể.

Bài tập 3 - Tam giác thần bí

- Cho một lưới ô vuông gồm n x n ô và số nguyên dương k. Tìm cách điền các số tự nhiên từ 1 đến 3n-3 vào các ô ở cột đầu tiên, dòng cuối cùng và đường chéo chính sao cho tổng các số điền trong cột đầu tiên, dòng cuối cùng và đường chéo chính của lưới đều bằng k. Ví dụ n=5, k = 35 ta có cách điền sau.

11				
10	3			
9		2		
1			7	
4	5	6	8	12

- Phát triển thuật toán dựa trên thuật toán quay lui để chỉ ra với giá trị của n, k cho trước bài toán có lời giải hay không. Nếu có câu trả lời chỉ cần đưa ra một lời giải.