Toán rời rạc

Phần thứ nhất

LÝ THUYẾT TỔ HỢP

Combinatorial Theory

Nội dung phần 1: Lý thuyết tổ hợp

Chương 1. Bài toán đếm

Chương 2. Bài toán tồn tại

Chương 3. Bài toán liệt kê tổ hợp

Chương 4. Bài toán tối ưu tổ hợp

Chương 3. BÀI TOÁN LIỆT KÊ TỔ HỢP

1. **Giới thiệu bài toán**
2. Thuật toán và độ phức tạp
3. Phương pháp sinh
4. Thuật toán quay lui

Giới thiệu bài toán

- Bài toán đưa ra danh sách tất cả cấu hình tổ hợp thoả mãn một số tính chất cho trước được gọi là ***bài toán liệt kê tổ hợp***.
- Do số lượng cấu hình tổ hợp cần liệt kê thường là rất lớn ngay cả khi kích thước cấu hình chưa lớn:
  - Số hoán vị của n phần tử là $n!$
  - Số tập con m phần tử của n phần tử là $n!/(m!(n-m)!)$

Do đó cần có quan niệm thế nào là giải bài toán liệt kê tổ hợp

Giới thiệu bài toán

- Bài toán liệt kê tổ hợp là giải được nếu như ta có thể *x*ác định một ***thuật toán*** để theo đó có thể lần lượt xây dựng được tất cả các cấu hình cần quan tâm.
- Một thuật toán liệt kê phải đảm bảo 2 yêu cầu cơ bản:
  - Không được lặp lại một cấu hình,
  - Không được bỏ sót một cấu hình.

Chương 3. BÀI TOÁN LIỆT KÊ TỔ HỢP

1. Giới thiệu bài toán
2. **Phương pháp sinh**
3. Thuật toán quay lui

3. Phương pháp sinh

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

3.1. Sơ đồ thuật toán

Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau được thực hiện:

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 đầu tiên và cấu hình cuối cùng trong thứ tự đã xác định.*

2) *Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có, đưa ra cấu hình kế tiếp nó.*

Thuật toán nói đến trong điều kiện 2) được gọi là Thuật toán *Sinh kế tiếp*

Thuật toán sinh

```
void Generate ( )
{
    <Xây dựng cấu hình đầu tiên>;
    stop = false;
    while (!stop)
    {
        <Đưa ra cấu hình đang có>;
        if (cấu hình đang có chưa là cuối cùng)
          <Sinh_kế_tiếp>;
        else stop = true;
    }
}
```

- <Sinh_kế_tiếp> là thủ tục thực hiện thuật toán sinh kế tiếp đã xây dựng trong điều kiện 2). Thủ tục này sẽ xây dựng cấu hình kế tiếp của cấu hình đang có trong thứ tự đã xác định.
- **Chú ý:** Do tập các cấu hình tổ hợp cần liệt kê là hữu hạn nên luôn có thể xác định được thứ tự trên nó. Tuy nhiên, thứ tự cần xác định sao cho có thể xây dựng được thuật toán Sinh kế tiếp.

3. Phương pháp sinh

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- **Sinh xâu nhị phân độ dài *n***
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

Sinh các xâu nhị phân độ dài *n*

Bài toán: Liệt kê tất cả các xâu nhị phân độ dài n:

$$b_1\, b_2 \ldots b_n, \text{ trong đó } b_i \in \{0, 1\}.$$

Giải:

- Thứ tự từ điển:

Xem mỗi xâu nhị phân $b = b_1\, b_2 \ldots b_n$ là biểu diễn nhị phân của một số nguyên $p(b)$

Ta nói xâu nhị phân $b = b_1\, b_2 \ldots b_n$ ***đi trước*** *xâu* nhị phân $b' = b'_1\, b'_2 \ldots b'_n$ trong thứ tự tự nhiên và ký hiệu là $b \square b'$ nếu $p(b) < p(b')$.

Sinh các xâu nhị phân độ dài *n*

Ví dụ: Khi n=3, các xâu nhị phân độ dài 3 được liệt kê theo thứ tự tự nhiên trong bảng bên

Dễ thấy thứ tự này trùng với thứ tự từ điển

b	*p*(*b*)
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Sinh các xâu nhị phân độ dài *n*

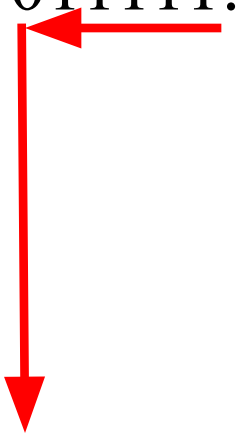
Thuật toán sinh kế tiếp:

- Xâu đầu tiên sẽ là 0 0 ... 0,
- Xâu cuối cùng là 1 1 ... 1.
- Giả sử $b_1\ b_2 \ldots b_n$ *là dãy đang có.*
  - Nếu xâu này gồm toàn số 1 □ kết thúc,
  - Trái lại, xâu kế tiếp nhận được bằng cách cộng thêm 1 (theo modul 2, có nhớ) vào xâu hiện tại.
- Từ đó ta có qui tắc sinh xâu kế tiếp như sau:
  - Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
  - Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

Sinh các xâu nhị phân độ dài *n*

- Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
- Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

Ví dụ: xét xâu nhị phân độ dài 10: $b =$ 1101011111.
Tìm xâu nhị phân kế tiếp.



- Ta có $i = 5$. Do đó, đặt:
  - $b_5 = 1$,
  - và $b_i = 0$, $i = 6, 7, 8, 9, 10$

□ ta thu đươc xâu nhị phân kế tiếp là 1101100000.

```
  1101011111
+
                1
  ----------------
  1101100000
```

Thuật toán sinh xâu kế tiếp

```
void  Next_Bit_String ( )
/*Sinh xâu nhị phân kế tiếp theo thứ tự từ điển của xâu
  đang có   b1 b2 ... bn ≠ 1 1 ... 1  */
{
     i=n;
    while  (bi == 1)
     {
      bi = 0;
   i=i-1;
    }
    bi = 1;
}
```

- Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
- Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

3. Phương pháp sinh

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- **Sinh tập con *m* phần tử của tập *n* phần tử**
- Sinh hoán vị của *n* phần tử

Sinh các tập con *m* phần tử của tập *n* phần tử

Bài toán đặt ra là: Cho X = = {1, 2, ... , *n*}. Hãy liệt kê các tập con *m* phần tử của X.

Giải:

- Thứ tự từ điển:

Mỗi tập con *m* phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm *m* thành phần

$$a = (a_1, a_2, \dots, a_m)$$

thỏa mãn

$$1 \le a_1 < a_2 < \dots < a_m \le n.$$

Sinh các tập con *m* phần tử của tập *n* phần tử

Ta nói tập con $a = (a_1, a_2, ..., a_m)$ đi trước tập con $a' = (a'_1, a'_2, ..., a'_m)$ trong thứ tự từ điển và kí hiệu là $a \square a'$, nếu tìm được chỉ số k ($1 \le k \le m$) sao cho:

$$a_1 = a'_1, a_2 = a'_2, \ldots, a_{k-1} = a'_{k-1},$$
$$a_k < a'_k .$$

Sinh các tập con *m* phần tử của tập *n* phần tử

Ví dụ: Các tập con 3 phần tử của $X = \{1, 2, 3, 4, 5\}$ 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

Sinh các tập con *m* phần tử của tập *n* phần tử

- **Thuật toán sinh kế tiếp:**
  - Tập con đầu tiên là (1, 2, ... , *m*)
  - Tập con cuối cùng là (*n*-*m*+1, *n*-*m*+2, ..., *n*).
  - Giả sử $a=(a_1, a_2, \dots, a_m)$ là tập con đang có chưa phải cuối cùng, khi đó tập con kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các quy tắc biến đổi sau đối với tập đang có:
    - Tìm từ bên phải dãy $a_1, a_2, \dots, a_m$ phần tử $a_i \neq n-m+i$
    - Thay a_i bởi $a_i + 1$
    - Thay a_j bởi $a_i + j - i$, với $j = i+1, i+2, \dots, m$

Sinh các tập con *m* phần tử của tập *n* phần tử

Ví dụ: $n = 6, m = 4$

Giả sử đang có tập con (1, 2, 5, 6), cần xây dựng tập con kế tiếp nó trong thứ tự từ điển.

- Tìm từ bên phải dãy $a_1, a_2, \ldots, a_m$ phần tử $a_i \neq n\text{-}m\text{+}i$
- Thay a_i bởi $a_i + 1$
- Thay a_j bởi $a_i + j - i$, với $j = i+1, i+2, \ldots, m$

- Ta có i=2:

Dãy:	(1, **2**, 5, 6)
Giá trị $n\text{-}m\text{+}i$:	(3, **4**, 5, 6)

thay $a_2 = a_2 + 1 = 3$

$a_3 = a_i + j - i = a_2 + 3 - 2 = 4$

$a_4 = a_i + j - i = a_2 + 4 - 2 = 5$

ta được tập con kế tiếp (1, 3, 4, 5).

Thuật toán sinh *m*-tập kế tiếp

```
void Next_Combination( )
/*Sinh tập con kế tiếp theo thứ tự từ điển của tập con
  (a1, a2,..., am)  ≠  (n-m+1,...,n) */
{
     i=m;
   while (ai == n-m+i)  i=i-1;
   ai= ai + 1;
   for  (j=i+1; j++; j <= m) aj = ai + j - i;
}
```

3. Phương pháp sinh

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- **Sinh hoán vị của *n* phần tử**

Sinh các hoán vị của tập *n* phần tử

Bài toán: Cho X = {1, 2, ... , *n*}, hãy liệt kê các hoán vị từ n phần tử của X.

Giải:

- Thứ tự từ điển:
  - Mỗi hoán vị từ *n* phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm n thành phần

$$a = (a_1, a_2, \dots, a_n)$$

thỏa mãn

$$a_i \in X, \quad i = 1, 2, \dots, n, \; a_p \neq a_q, \; p \neq q.$$

Sinh các hoán vị của tập *n* phần tử

Ta nói hoán vị $a = (a_1, a_2, ..., a_n)$ đi trước hoán vị $a' = (a'_1, a'_2, ..., a'_n)$ trong thứ tự từ điển và ký hiệu là $a \square a'$, nếu tìm được chỉ số k ($1 \le k \le n$) sao cho:

$$a_1 = a'_1, a_2 = a'_2, ..., a_{k-1} = a'_{k-1},$$

$$a_k < a'_k.$$

Sinh các hoán vị của tập *n* phần tử

- Ví dụ: Các hoán vị từ 3 phần tử của X ={1, 2, 3} được liêt 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

Sinh các hoán vị của tập *n* phần tử

- **Thuật toán sinh kế tiếp:**
  - Hoán vị đầu tiên: $(1, 2, \ldots, n)$
  - Hoán vị cuối cùng: $(n, n\text{-}1, \ldots, 1)$.
  - Giả sử $a = (a_1, a_2, \ldots, a_n)$ là hoán vị chưa phải cuối cùng, khi đó hoán vị kế tiếp nó có thể xây dựng nhờ thực hiện các biến đổi sau:
    - Tìm từ phải qua trái: chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
    - 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 .

Sinh các hoán vị của tập *n* phần tử

Ví dụ: Giả sử đang có hoán vị (3, 6, 2, 5, 4, 1), cần xây dựng hoán vị kế tiếp nó trong thứ tự từ điển.

- Tìm từ phải qua trái: chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
- 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

- Ta có chỉ số $j = 3$ ($a_3 = 2 < a_4 = 5$).
- Số nhỏ nhất còn lớn hơn a_3 trong các số bên phải của a_3 là $a_5 = 4$. Đổi chỗ a_3 với a_5 trong hoán vị đã cho (3, 6, **2**, 5, **4**, 1)

  ta thu được (3, 6, **4**, 5, **2**, 1),
- Cuối cùng, lật ngược thứ tự đoạn $a_4\ a_5\ a_6$ ta thu được hoán vị kế tiếp (3, 6, 4, **1, 2, 5**).

Thuật toán sinh hoán vị kế tiếp

```
void  Next_Permutation ( )
{
   /*Sinh hoán vị kế tiếp (a1, a2, ..., an) ≠ (n, n-1, ..., 1)   */
   // Tìm j là chỉ số lớn nhất thoả mãn aj < aj+1 :
    j=n-1;  while (aj > aj+1 ) j =j-1;
  // Tìm ak là số nhỏ nhất còn lớn hơn aj ở bên phải aj :
    k=n;  while (aj > ak ) k=k-1;
    Swap(aj, ak); //đổi chỗ aj với ak
    // Lật ngược đoạn từ aj+1 đến an :
    r=n; s=j+1;
    while ( r>s )
  {
      Swap(ar, as); // đổi chỗ ar với as
       r=r-1; s= s+1;
    }
}
```

Chương 3. BÀI TOÁN LIỆT KÊ TỔ HỢP

1. Giới thiệu bài toán
2. Phương pháp sinh
3. **Thuật toán quay lui**

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài n
- Liệt kê tập con m phần tử của tập n phần tử
- Liệt kê hoán vị

Sơ đồ thuật toán quay lui

- ***Bài toán liệt kê (Q):*** *Cho $A_1, A_2, ..., A_n$ là các tập hữu hạn. Ký hiệu*

$$A = A_1 \times A_2 \times ... \times A_n = \{ (a_1, a_2, ..., a_n): \ a_i \in A_i, \ i=1, 2, ..., n\}.$$

Giả sử P là tính chất cho trên A. Vấn đề đặt ra là liệt kê tất cả các phần tử của A thoả mãn tính chất P:

$$D = \{ a = (a_1, a_2, ..., a_n) \in A: \ a \text{ thoả mãn tính chất } P \}.$$

- Các phần tử của tập *D* được gọi là các ***lời giải chấp nhận được.***

Sơ đồ thuật toán quay lui

Tất cả các bài toán liệt kê tổ hợp cơ bản đều có thể phát biểu dưới dạng bài toán liệt kê (Q).

Ví dụ:

- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

$$B^n = \{(a_1, ..., a_n): a_i \in \{0, 1\}, i=1, 2, ..., n\}.$$

- Bài toán liệt kê các tập con m phần tử của tập $N = \{1, 2, ..., n\}$ đòi hỏi liệt kê các phần tử của tập:

$$S(m,n) = \{(a_1,..., a_m) \in N^m: 1 \le a_1 < ... < a_m \le n\}.$$

- TËp c¸c ho¸n vÞ cña c¸c sè tù nhi^n 1, 2, ..., n lµ tËp

$$\Pi_n = \{(a_1,..., a_n) \in N^n: a_i \ne a_j; i \ne j\}.$$

Lời giải bộ phận

Lời giải của bài toán là bộ có thứ tự gồm n thành phần $(a_1, a_2, ..., a_n)$, trong đó $a_i \in A_i$, $i = 1, 2, ..., n$.

Định nghĩa. Ta gọi lời giải bộ phận cấp k $(0 \le k \le n)$ là bộ có thứ tự gồm k thành phần

$$(a_1, a_2, ..., a_k),$$

trong đó $a_i \in A_i$, $i = 1, 2, ..., k$.

- Khi $k = 0$, lời giải bộ phận cấp 0 được ký hiệu là ( ) và còn được gọi là lời giải rỗng.
- Nếu $k = n$, ta có lời giải đầy đủ hay đơn giản là một lời giải của bài toán.

Sơ đồ thuật toán quay lui

Thuật toán quay lui được xây dựng dựa trên việc xây dựng dần từng thành phần của lời giải.

- Thuật toán bắt đầu từ lời giải rỗng ( ).
- Trên cơ sở tính chất P ta xác định được những phần tử nào của tập A_1 có thể chọn vào vị trí thứ nhất của lời giải. Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ nhất của lời giải. Ký hiệu tập các UCV vào vị trí thứ nhất của lời giải là S_1. Lấy $a_1 \in S_1$, bổ sung nó vào lời giải rỗng đang có ta thu được lời giải bộ phận cấp 1: (a_1).

Sơ đồ thuật toán quay lui

- Bước tổng quát: Giả sử ta đang có lời giải bộ phận cấp k-1:

$$(a_1, a_2, ..., a_{k-1}),$$

cần xây dựng lời giải bộ phận cấp k:

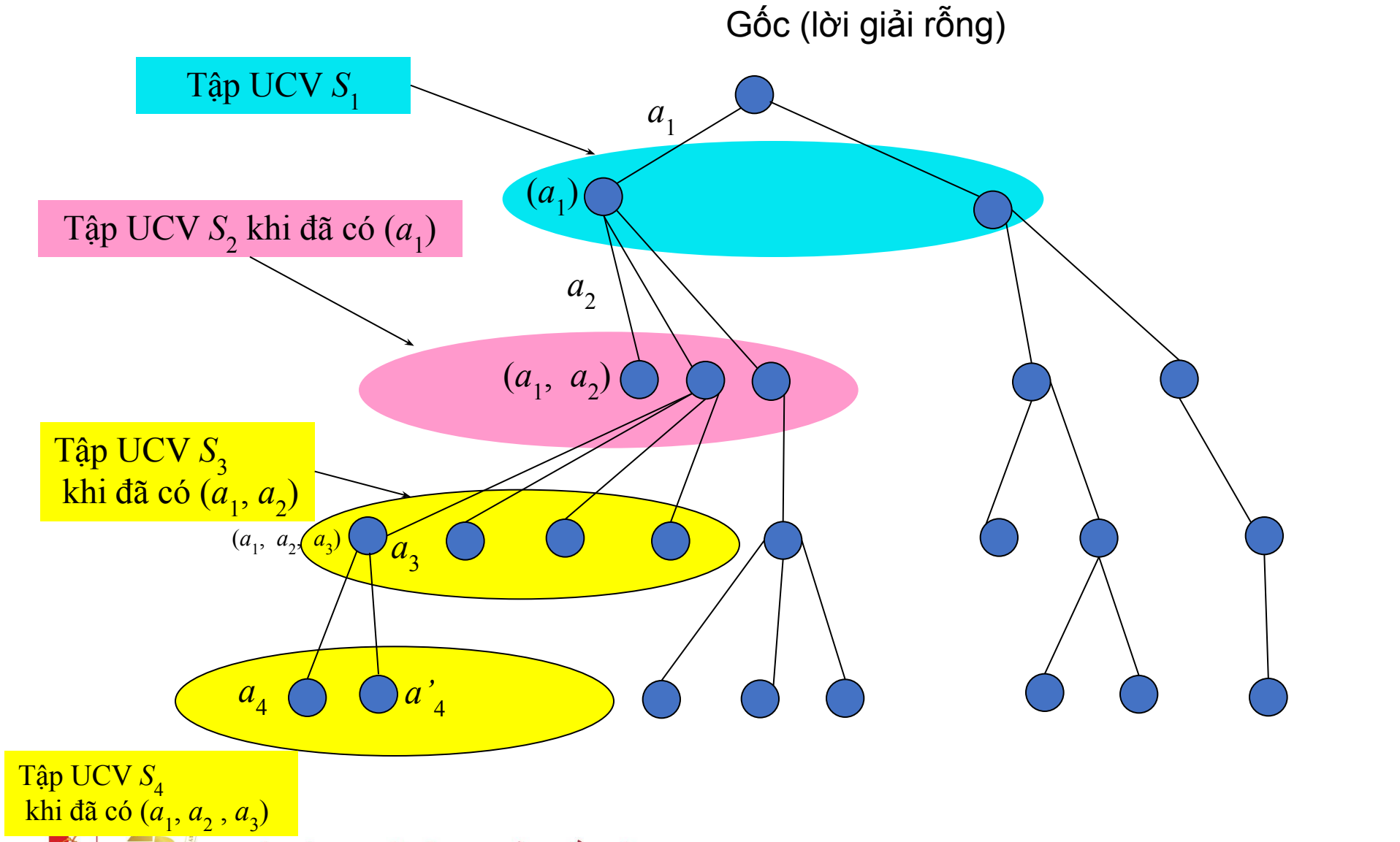
$$(a_1, a_2, ..., a_{k-1}, \mathbf{a_k})$$

  - Trên cơ sở tính chất P, ta xác định được những phần tử nào của tập A_k có thể chọn vào vị trí thứ k của lời giải.
  - Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ k của lời giải khi k-1 thành phần đầu của nó đã được chọn là $(a_1, a_2, ..., a_{k-1})$. Ký hiệu tập các ứng cử viên này là S_k.
  - Xét 2 tình huống:
    - $S_k \neq \varnothing$
    - $S_k = \varnothing$

Sơ đồ thuật toán quay lui

- **$S_k \neq \varnothing$:** Lấy $a_k \in S_k$ bổ sung nó vào lời giải bộ phận cấp k-1 đang có (a_1, a_2, ..., a_{k-1}) ta thu được lời giải bộ phận cấp k (a_1, a_2, ..., a_{k-1}, a_k). Khi đó
  - Nếu $k = n$ thì ta thu được một lời giải,
  - Nếu $k < n$, ta tiếp tục đi xây dựng thành phần thứ k+1 của lời giải.
- **$S_k = \varnothing$:** Điều đó có nghĩa là lời giải bộ phận (a_1, a_2, ..., a_{k-1}) không thể tiếp tục phát triển thành lời giải đầy đủ. Trong tình huống này ta **quay trở lại** tìm ứng cử viên mới vào vị trí thứ k-1 của lời giải (chú ý: ứng cử viên mới này nằm trong tập S_{k-1})
  - Nếu tìm thấy UCV như vậy, thì bổ sung nó vào vị trí thứ k-1 rồi lại tiếp tục đi xây dựng thành phần thứ k.
  - Nếu không tìm được thì ta lại **quay trở lại** thêm một bước nữa tìm UCV mới vào vị trí thứ k-2, ... Nếu quay lại tận lời giải rỗng mà vẫn không tìm được UCV mới vào vị trí thứ 1, thì thuật toán kết thúc.

Cây liệt kê lời giải theo thuật toán quay lui



Thuật toán quay lui (đệ qui)

```
void Try(int k)
{
  <Xây dựng Sk là tập chứa các ứng cử
  viên cho vị trí thứ k của lời giải>;
  for y ∈ Sk  //Với mỗi UCV y từ Sk
  {
      ak = y;
    if (k == n) then <Ghi nhận lời
            giải (a1, a2, ..., ak) >;
      else Try(k+1);
    Trả các biến về trạng thái cũ;
  }
}
```

Lệnh gọi để thực hiện thuật toán quay lui là:

```
Try(1);
```

- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Try(1) sau khi ghi nhận được lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.

Thuật toán quay lui (không đệ qui)

```
void Backtracking ( )
{
  k=1;
  <Xây dựng Sk>;
  while (k > 0)  {
    while (Sk ≠ ∅ ) {
      ak ← Sk; // Lấy ak từ Sk
      if  <(k == n) >  then
        <Ghi nhận (a1,a2,...,ak) >;
      else {
          k = k+1;
          <Xây dựng Sk>;
      }
    }
    k = k - 1;    // Quay lui
  }
}
```

Lệnh gọi để thực hiện thuật toán quay lui là:

```
Backtracking ( );
```

Hai vấn đề mấu chốt

- Để cài đặt thuật toán quay lui giải các bài toán tổ hợp cụ thể ta cần giải quyết hai vấn đề cơ bản sau:
  - Tìm thuật toán xây dựng các tập UCV S_k
  - Tìm cách mô tả các tập này để có thể cài đặt thao tác liệt kê các phần tử của chúng (cài đặt vòng lặp qui ước **for** $y \in S_k$ **).**
- Hiệu quả của thuật toán liệt kê phụ thuộc vào việc ta có xác định được chính xác các tập UCV này hay không.

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- **Liệt kê xâu nhị phân độ dài *n***
- Liệt kê tập con *m* phần tử của tập *n* phần tử
- Liệt kê hoán vị

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

- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

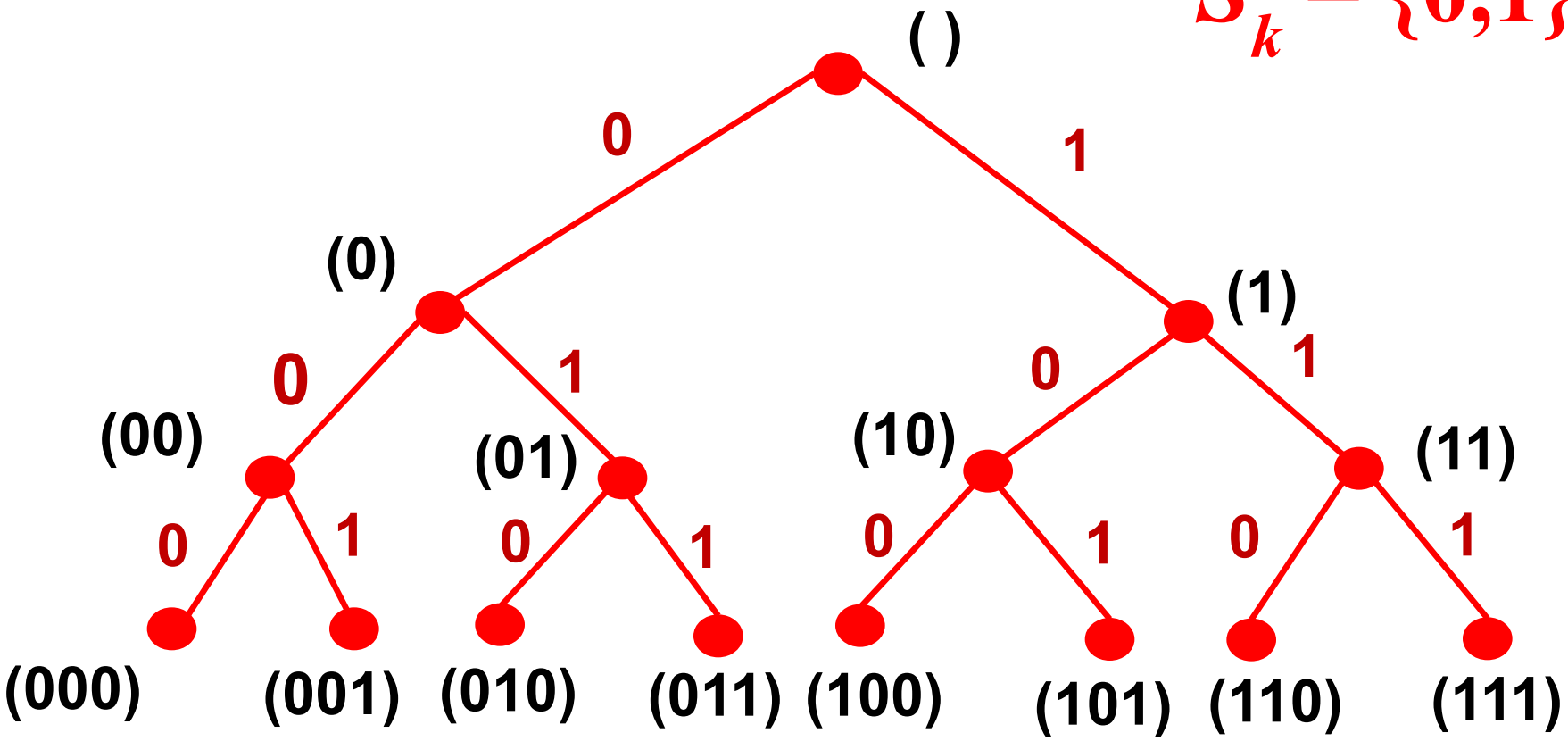
$$B^n = \{(b_1, ..., b_n): b_i \in \{0, 1\}, i=1, 2, ..., n\}.$$

- Ta xét cách giải quyết hai vấn đề cơ bản để cài đặt thuật toán quay lui:
  - Xây dựng tập ứng cử viên S_k: Rõ ràng ta có $S_1 = \{0, 1\}$. Giả sử đã có xâu nhị phân cấp k-1 $(b_1, ..., b_{k-1})$, khi đó rõ ràng $S_k = \{0,1\}$. Như vậy, tập các UCV vào các vị trí của lời giải đã được xác định.
  - Cài đặt vòng lặp liệt kê các phần tử của S_k: ta có thể sử dụng vòng lặp for

```
for (y=0;y<=1;y++)
```

C©y liÖt kª d·y nhÞ ph©n ®é dµi 3

$S_k = \{0,1\}$

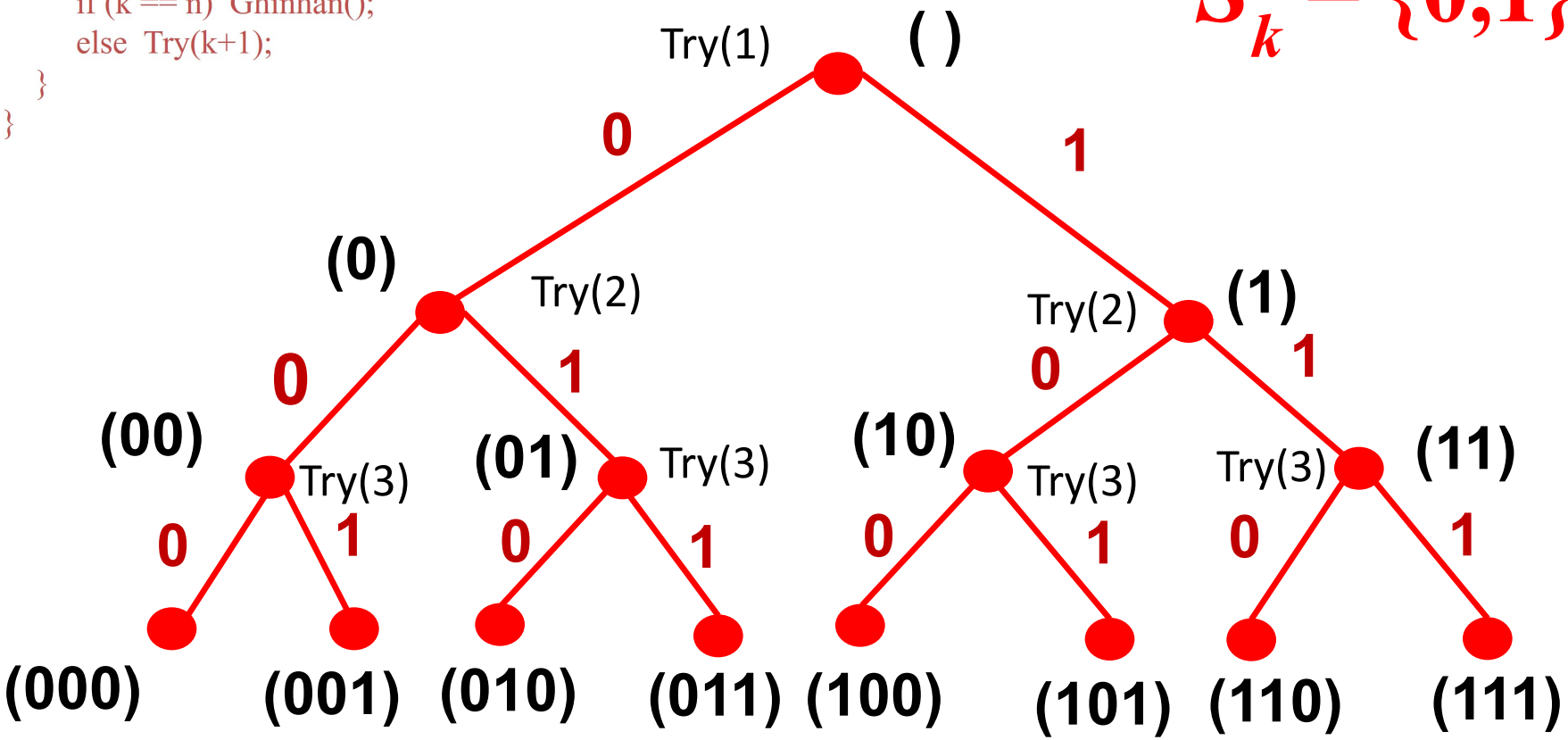


NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

C©y liÖt kª d·y nhÞ ph©n ®é dµi 3

```
void  Try(int k){
  for (int j = 0; j<=1; j++) {
      b[k] = j;
      if (k == n)  Ghinhan();
      else  Try(k+1);
   }
}
```

$S_k = \{0,1\}$



Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài *n*
- **Liệt kê tập con *m* phần tử của tập *n* phần tử**
- Liệt kê hoán vị

Ví dụ 2. Liệt kê các *m*-tập con của *n*-tập

Bài toán: Liệt kê các tập con *m* phần tử của tập $N = \{1, 2, ..., n\}$.

Ví dụ: Liệt kê các tập con 3 phần tử của tập N = {1, 2, 3, 4, 5}

Giải: (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)

- Bài toán dẫn về: Liệt kê các phần tử của tập:

$$S(m,n)=\{(a_1,..., a_m)\in N^m: 1 \le a_1<...<a_m \le n\}$$

Ví dụ 2. Liệt kê các *m*-tập con của *n*-tập

Ta xét cách giải quyết 2 vấn đề cơ bản để cài đặt thuật toán quay lui:

- Xây dựng tập ứng cử viên S_k:
  - Từ điều kiện: $1 \le a_1 < a_2 < ... < a_m \le n$

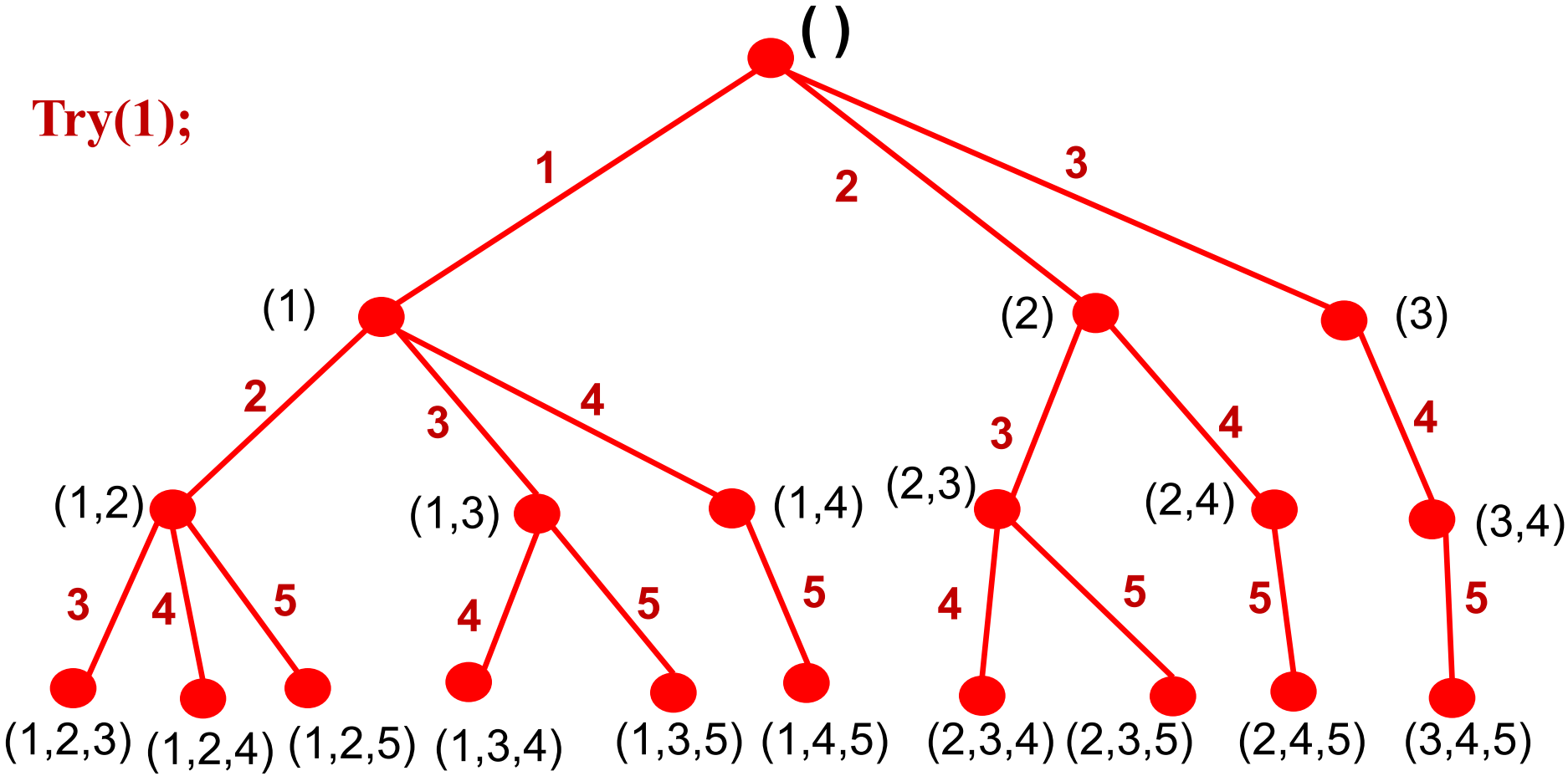
    suy ra $S_1 = \{1, 2, ..., n-(m-1)\}$.
  - Giả sử có tập con $(a_1, ..., a_{k-1})$. Từ điều kiện $a_{k-1} < a_k < ... < a_m \le n$, ta suy ra:

    $S_k = \{a_{k-1}+1, a_{k-1}+2, ..., n-(m-k)\}$.
- Cài đặt vòng lặp liệt kê các phần tử của S_k:

```
for (y=a[k-1]+1;y<=n-m+k;y++) ...
```

Cây liệt kê S(5,3)

Try(1);



$$S_k = \{a_{k-1}+1, a_{k-1}+2, ..., n-(m-k)\}$$

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài *n*
- Liệt kê tập con *m* phần tử của tập *n* phần tử
- **Liệt kê hoán vị**

Ví dụ 3. Liệt kê hoán vị

Tập các hoán vị của các số tự nhiên 1, 2, ..., n là tập:

$$\Pi_n = \{(x_1,..., x_n) \in N^n: \ x_i \neq x_j, \ i \neq j \}.$$

Bài toán: Liệt kê tất cả các phần tử của Π_n

Ví dụ 3. Liệt kê hoán vị

- Xây dựng tập ứng cử viên S_k:
  - Râ rµng $S_1 = N$. Gi¶ sö ta ®ang cã ho¸n vÞ bé phËn $(a_1, a_2, ..., a_{k-1})$, tõ ®iÒu kiÖn $a_i \neq a_j$, víi mäi $i \neq j$ ta suy ra

$$S_k = N \setminus \{ a_1, a_2, ..., a_{k-1} \}.$$

Mô tả S_k

Xây dựng hàm nhận biết UCV:

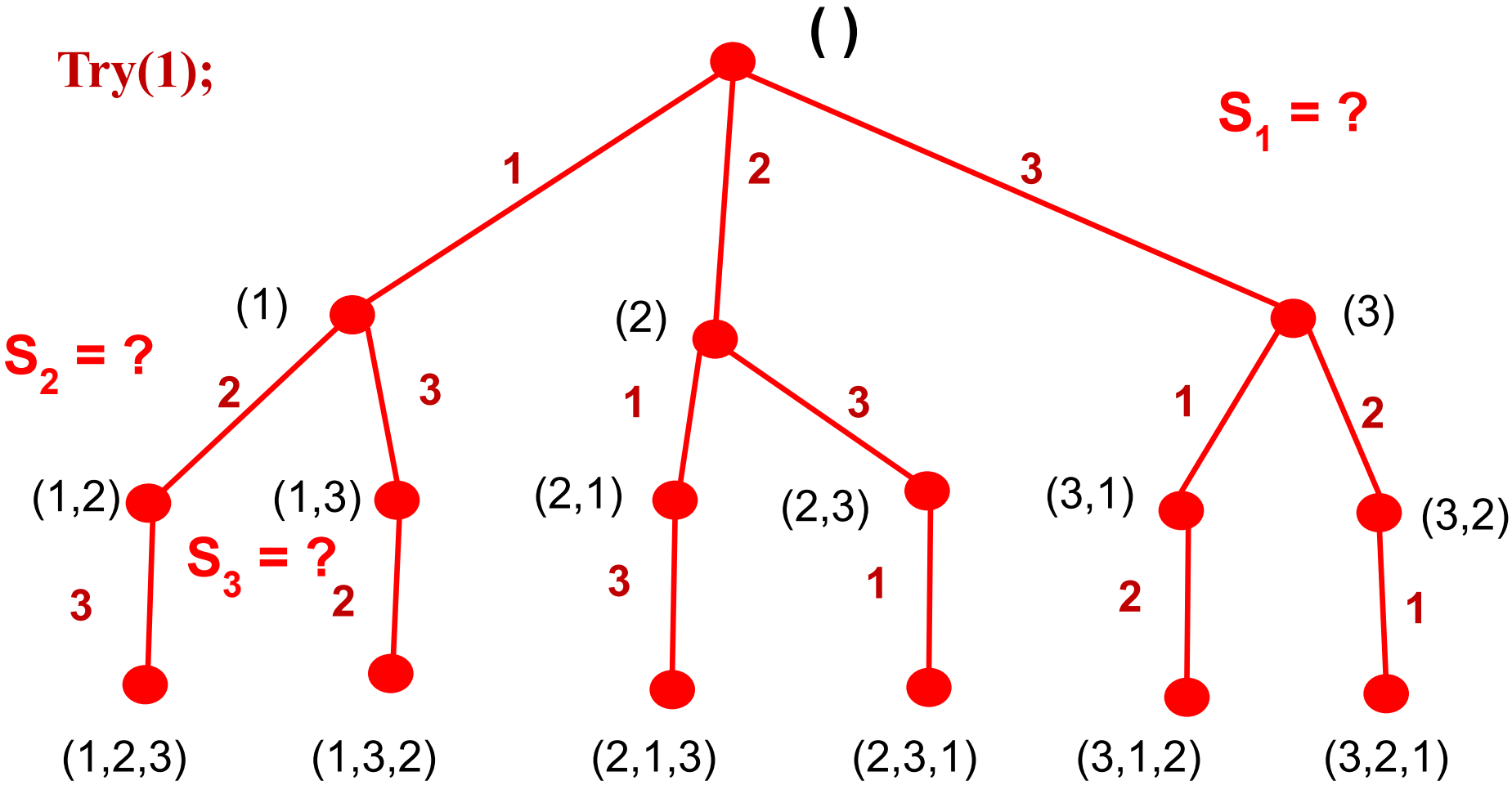
```
bool UCV(int j, int k)
{
  //UCV nhận giá trị true khi và chỉ khi j ∈ Sk
    int i;
    for (i=1;i++;i<=k-1)
        if (j == a[i]) return false;
    return true;
}
```

Liệt kê hoán vị

- Cài đặt vòng lặp liệt kê các phần tử của S_k:

```
for (y=1; y++; y <= n)
    if  (UCV(y, k) )
    {
        // y là UCV vào vị trí k
        . . .
    }
```

Cây liệt kê hoán vị của 1, 2, 3



$$S_k = N \setminus \{ a_1, a_2, ..., a_{k-1} \}$$