TOÁN RỜI RẠC 1

CHƯƠNG 3

Giảng viên: Vũ Văn Thỏa

CHƯƠNG 3. BÀI TOÁN LIỆT KÊ TỔ HỢP

- Bài toán liệt kê tổ hợp
- Phương pháp sinh liệt kê tổ hợp
- Phương pháp quay lui liệt kê tổ hợp

3.1 Bài toán liệt kê tổ hợp

- Mỗi hoán vị hoặc tổ hợp cụ thể gọi là một cấu hình tổ hợp. Bài toán liệt kê tổ hợp là đưa ra danh sách tất cả các cấu hình tổ hợp có thể có.
- Hai nguyên tắc liệt kê:
  + Không bỏ sót cấu hình
  + Không lặp lại cấu hình

- Mọi tập S có n phần tử có thể đánh số các phần tử theo các chỉ số i = 1, 2, ..., n.

$\Rightarrow$ có sự tương ứng một - một giữa tập S với tập n số nguyên dương nhỏ nhất {1, 2, ..., n}.

$\Rightarrow$ liệt kê tất cả các hoán vị hay tổ hợp chập k phần tử của tập S cũng tương đương với việc liệt kê tất cả các hoán vị hay tổ hợp chập k phần tử của tập {1, 2, ..., n}.

3.2 Thuật toán sinh liệt kê tổ hợp

- Xác định một thứ tự trên tập các cấu hình tổ hợp. Từ đó xác định cấu hình đầu tiên và cấu hình cuối cùng.
- Xây dựng thuật toán đưa ra cấu hình kế tiếp từ cấu hình hiện có chưa phải là cấu hình cuối cùng

(1) Thuật toán sinh hoán vị kế tiếp

- Thứ tự trên tập các hoán vị: thứ tự từ điển

$\Rightarrow$ Hoán vị đầu tiên: (1, 2, …, n-1, n)

$\Rightarrow$ Hoán vị cuối cùng: (n, n-1, …, 2, 1)

- **Thuật toán sinh kế tiếp:**

Input:

n;

Output:

Tất cả các hoán vị a[1], …, a[n];

Ý tưởng:

- Khởi tạo: a[i] = i, 1 ≤ i ≤ n
- Quá trình lặp:

- Chọn i lớn nhất sao cho a[i] < a[i+1]:

(a[i] < a[i+1]) ∧ (a[i+1] > a[i+2] > ... > a[n]);

Nếu không tìm được i thì kết thúc;

- Tìm k > i sao cho giá trị a[k] bé nhất trong số các giá trị > a[i] rồi đổi chỗ a[i] và a[k];

- Lật ngược đoạn từ vị trí thứ i+1 đến vị trí n;

Biểu diễn thuật toán:

```
void HvKe(int a[], int n){
  for (int i = 1; i <= n; i++)
                    a[i] = i;
  while (1) {
        for (i = 1; i <= n; i++)
                  cout << a[i] << " ";
       cout << endl;
    i = n-1;
    while (i > 0 && a[i] > a[i+1])
                  i--;
     if (i == 0) return;
        //Da o hoan vi cuoi cung
int k = n;
    while (a[k] < a[i]) k--;
   int tmp = a[i];
        a[i] = a[k]; a[k] = tmp;
    int l= i+1; int r = n;
    while (l < r)  {
            tmp = a[l]; a[l] = a[r];
     a[r] = tmp;
            l++; r--;   }
    }
}
```

Test thuật toán:

- Cho n = 5; a[] = (1, 2, 4, 5, 3)
- Tìm i = 3; a[i] = 4;
- Tìm k = 4; a[k] = 5;
- Đổi chỗ a[i] ↔ a[k]: (1, 2, **5**, **4**, 3)
- Lật ngược đoạn từ i+1 = 4 đến n = 5:

  (1, 2, 5, 3, 4)
- Hoán vị kế tiếp: a[] = (1, 2, 5, 3, 4)

(2) Thuật toán sinh tổ hợp kế tiếp

- Thứ tự trên tập các tổ hợp: thứ tự từ điển

$\Rightarrow$ Tổ hợp đầu tiên: (1, 2, …, k)

$\Rightarrow$ Tổ hợp cuối cùng: (n- k + 1, …, n-1, n)

- **Thuật toán sinh kế tiếp:**

Input: n, k;

Output: Tất cả các tổ hợp a[1], …, a[k];

Ý tưởng:

- Khởi tạo: a[i] = i, 1 ≤ i ≤ k;
- Quá trình lặp:

- Chọn i lớn nhất sao cho a[i] < n – k + i;
  Nếu không tìm được i thì kết thúc;
- Thay thế a[i] = a[i] + 1;
- Tính a[j] = a[i] (mới) + j - i với j = i+1, i+2, ..., k ;

Biểu diễn thuật toán:

```
void ThKe(int a[],int n,int k){
  for (int i = 1; i <= k; i++)
                    a[i] = i;
  while (1) {
     for (i = 1; i <= k; i++)
              cout << a[i] << " " ;
      cout << endl;
     i = k;
 while (i > 0 && a[i] >= n-k+i)
                      i--;
if (i == 0) return;//Da o to hop
     cuoi cung;
     a[i] = a[i] + 1;
     for (int j = i+1; j <= k; j++)
       a[j] = a[i] + j - i;
       }
    }
```

Test thuật toán:

- Cho n = 5, k = 3; a[] = (1, 4, 5)
- Tổ hợp cuối cùng: (3, 4, 5)
- Tìm i = 1; a[i] = 1;
- Thay thế a[1] = 1 + 1 = 2;
- Tính:

a[2] = 2 + 2 – 1 = 3; a[3] = 2 + 3 – 1 = 4;

- Tổ hợp kế tiếp: a[] = (2, 3, 4)

(3) Thuật toán sinh xâu nhị phân kế tiếp:

- Thứ tự trên tập xâu nhị phân: thứ tự từ điển

$\Rightarrow$ Xâu nhị phân đầu tiên: 00…0

$\Rightarrow$ Xâu nhị phân cuối cùng: 11…1

Thuật toán sinh kế tiếp:

- **Input:**

  n;

- **Output:**

  Tất cả các xâu nhị phân:

  x[1], x[2], …, x[n];

Ý tưởng:

- Khởi tạo: x[i] = 0, 1 ≤ i ≤ n;
- Quá trình lặp:

- Chọn i lớn nhất sao cho x[i] = 0;
  Nếu không tìm được i thì kết thúc;
- Thay thế x[i] = 1;
- Thay thế x[j] = 0 với j = i+1, i+2, ..., n ;

Biểu diễn thuật toán:

```
void XnpKt(int x[], int n) {
   for (int i = 1; i<= n; i++) x[i] = 0;
   while (1) {
    for (i = 1; i <= n; i++) cout << x[i];
    cout << endl;
    i = n;
    while (i > 0 && x[i] == 1) {x[i] = 0; i--;}
    if (i == 0) return; else x[i] = 1;}
   }
```

Test thuật toán:

- n = 10, x[] = (1, 0, 1, 0, 1, 0, 1, 0, 1, 1)
- Tìm i lớn nhất sao cho x[i] = 0: i = 8
- Giữ nguyên giá trị của x[j] với $1 \leq j \leq 7$
- Thay thế x[8] = 1 và x[j] = 0 với $9 \leq j \leq 10$
- Xâu nhị phân kế tiếp:

x[] = (1, 0, 1, 0, 1, 0, 1, 1, 0, 0)

3.3 Thuật toán quay lui liệt kê tổ hợp

- Thuật toán quay lui liệt kê các hoán vị
- Thuật toán quay lui liệt kê các tổ hợp
- Thuật toán quay lui liệt kê các xâu nhị phân

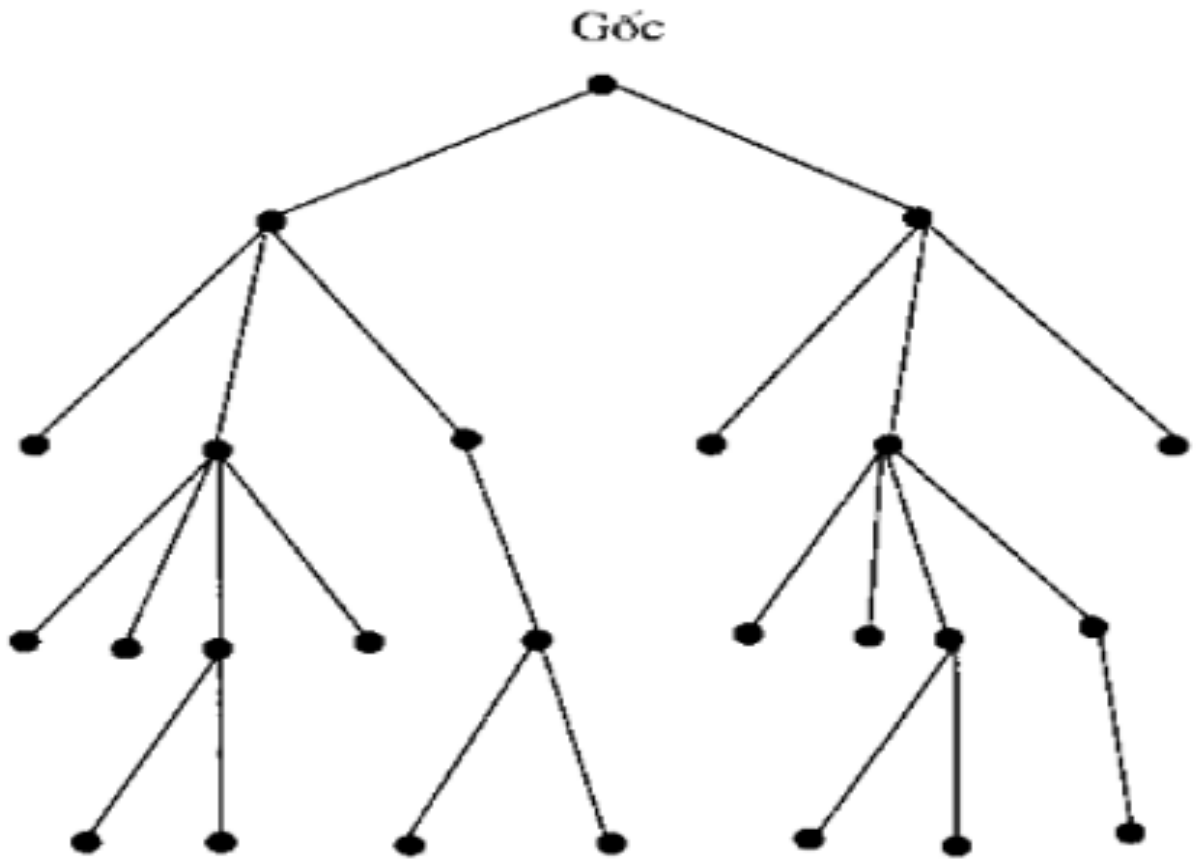
Giới thiệu phương pháp quay lui

- **Ý tưởng phương pháp quay lui (*Backtracking*)**
  - Dùng để giải bài toán liệt kê các cấu hình $X = (x_1, \ldots, x_n)$.
  - Mỗi X được xây dựng từng thành phần x_m ($1 \leq m \leq n$).
  - Giả sử đã xây dựng được m-1 thành phần $x_1, \ldots, x_{m-1}$:
  - ✓ Thành phần x_m được chọn bằng cách thử tất cả các khả năng $x_m = i \in D_m$. Nếu chọn được x_m tiếp tục xây dựng thành phần x_{m+1}
  - ✓ Nếu không chọn được x_m thì quay lui chọn lại x_{m-1}
  - Khi xây dựng đủ n thành phần sẽ thông báo/ghi nhận cấu hình X nhận được.
- *Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu DFS (Depth-First Search).*

Thuật toán quay lui xây dựng thành phần thứ m

```
Backtracking(m) {
        for (Mỗi phương án chọn i ∈ Di) {
                if (Chấp nhận i) {
                        <Chọn i cho X[m]]>;
                        if (Chọn đủ n thành phần) {
                                <Đưa ra cấu hình X>;
                        } else {
                                Backtracking(m+1);
                                <Bỏ chọn i cho X[m]]>;
                            }
                    }
                }
         }
```

Mô hình cây tìm kiếm sử dụng quay lui



Khả năng chọn x_1

Khả năng chọn x_2 với x_1 đã chọn

Khả năng chọn x_3 với x_1, x_2 đã chọn

Ứng dụng cho liệt kê tổ hợp

- Xây dựng từng thành phần thứ m của cấu hình tổ hợp;
- Khi đã đủ các thành phần $\Rightarrow$ Kết thúc 1 lần lặp và ghi nhận thêm 1 cấu hình tổ hợp;
- Cài đặt bằng đệ quy:

- *Tạo hàm đệ quy xây dựng thành phần thứ m của cấu hình tổ hợp;*
- *Trong hàm liệt kê tổ hợp sử dụng hàm đệ quy xây dựng các thành phần của cấu hình tổ hợp bắt đầu từ thành phần thứ 1.*

Thuật toán quay lui liệt kê các hoán vị:

- Input:

  n;

- Output:

  Tất cả các hoán vị dạng:

  a[1], a[2], …, a[n];

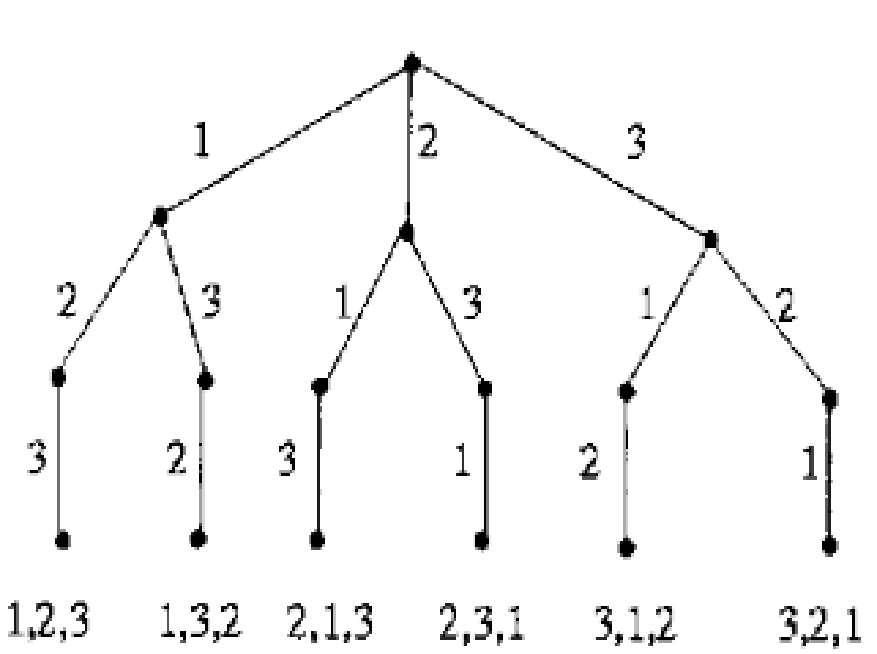
Biểu diễn thuật toán:

```
int vs[100], a[100], n;
void HvDq(int m) {
   if (m > n) {
      for int i = 1; i <= n; i++)
            cout << a[i] << " ";
       cout << endl; return;}
    for (i = 1; i <= n; i++)
      if (vs[i] == 0){
         a[m] = i; vs[i] = 1;
   HvDq(m+1); vs[i] = 0;}
   }
```

```
void LkDqHv(int a[], int n){
    for (int i = 1; i <= n; i++)
                vs[i] = 0;
    HvDq(1);
}
```

Test thuật toán liệt kê hoán vị sử dụng quay lui:

- Với n = 3, quá trình liệt kê các hoán vị sử dụng thuật toán quay lui tạo thành cây tìm kiếm sau:



Thuật toán quay lui liệt kê các tổ hợp:

- **Input**:

  n, k;

- **Output**:

  Tất cả các tổ hợp dạng: a[1], a[2], …, a[k];

- **Ghi chú**:

➢ Trong tổ hợp quy ước: a[1] < a[2] < … < a[k]

➢ Phạm vi của a[m]:

$$a[m-1] + 1 \leq a[m] \leq n - (k-m) = n + m - k$$

Biểu diễn thuật toán:

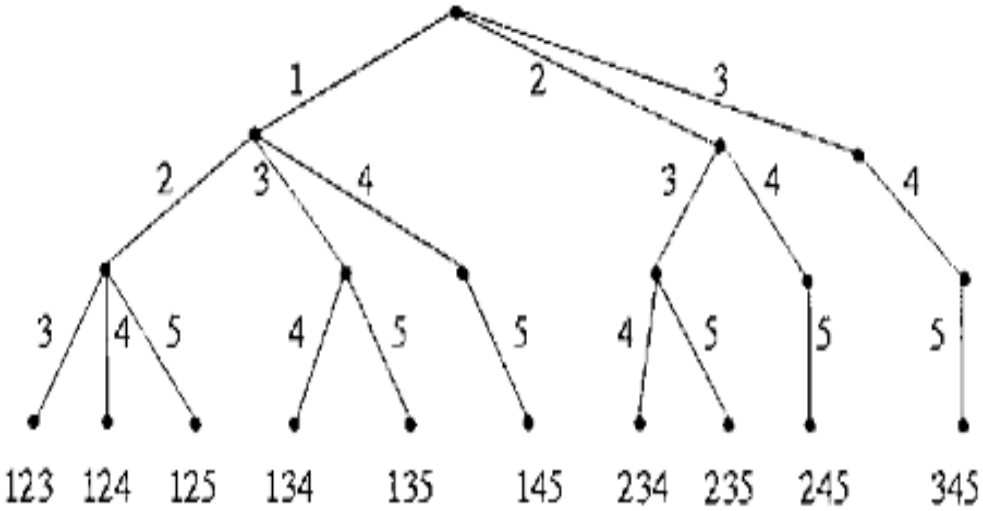
```
int a[100], n, k;

void ThDq(int m) {
   if (m > k){
       for int i = 1; i <= k; i++)
           cout << a[i] << " ";
        cout << endl; return;}
    for (i = a[m-1] + 1; i <= n-
    k+m; i++){
           a[m] = i;
          ThDq(m+1); }
   }
```

```
void LkDqTh(int a[],int n, int k){
      ThDq(1);
}
```

Test thuật toán liệt kê tổ hợp sử dụng quay lui:

- Với n = 5, k = 3, quá trình liệt kê các tổ hợp sử dụng thuật toán quay lui tạo thành cây tìm kiếm sau:



Thuật toán quay lui liệt kê các xâu nhị phân:

- Input:

  n;

- Output:

  Tất cả các xâu nhị phân dạng:

  x[1], x[2], …, x[n];

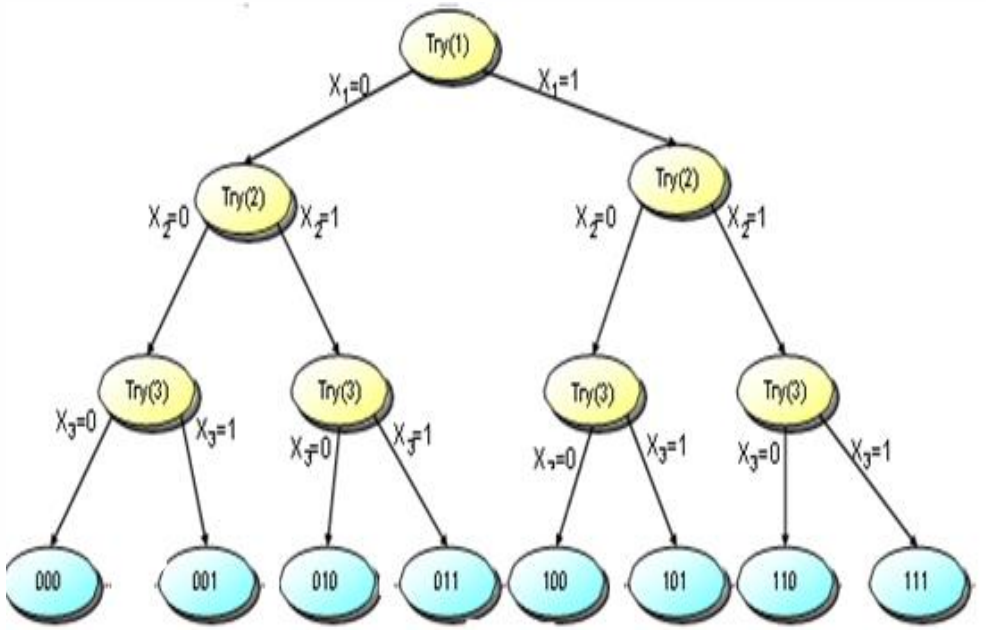
Biểu diễn thuật toán:

```
int  x[100], n;
void XnDq(int m) {
   if (m > n) {
     for int i = 1; i <= n; i++)
              cout << x[i];
     cout << endl; return;}
    for (int i = 0; i <= 1; i++){
            x[m] = i;
    XnDq(m+1); }
   }
```

```
void LkDqXn(int x[], int n){
      XnDq(1);
}
```

Test thuật toán liệt kê xâu nhị phân sử dụng quay lui:

- Với n = 3, quá trình liệt kê các xâu nhị phân sử dụng thuật toán quay lui tạo thành cây tìm kiếm sau:



Ghi chú

- Các hoán vị, các tổ hợp chập k của n phần từ và các xâu nhị phân liệt kê bằng phương pháp quay lui cũng xuất hiện theo thứ tự từ điển.
- Kết quả liệt kê tổ hợp bằng phương pháp sinh kế tiếp và bằng quay lui là như nhau.
- Tốc độ chạy chương trình cài đặt bằng đệ quy sẽ nhanh hơn.
- Chương trình cài đặt bằng phương pháp sinh dễ gỡ lỗi và dễ hiểu hơn.

Thảo luận


THANK YOU