Thuật Toán Sinh

1. 🎯 Giới thiệu

Thuật toán sinh (*generative algorithm*) là một phương pháp vét cạn được dùng với các bài toán liệt kê hoặc đếmcấuhình, thỏa các yêu cầu sau:

- Có thể xác định được cấu hình đầu tiên và cấu hình cuối cùng
- Tìm được thuật toán để từ cấu hình hiện tại sinh ra cấu hình kế tiếp

- Thuật toán **sinh** (generation algorithm) được áp dụng trong **các bài toán liệt kê tổ hợp, hoán vị, tập con...**, tức là các bài toán cần **xét toàn bộ không gian cấu hình** để tìm lời giải thỏa mãn điều kiện. Cụ thể, nó được dùng trong các vấn đề sau:

---

2. 🧩 Mã giả chung

```
SinhCauHinh() {
  <Bước 1> : Khởi tạo
      <Khởi tạo cấu hình đầu tiên>
   <Bước 2> : Lặp
   while(<Chưa gặp cấu hình cuối cùng>){
      <Đưa ra cấu hình hiện tại>
      <Sinh ra cấu hình kế tiếp>
   }
   <Đưa ra cấu hình cuối cùng
}
```

Tùy theo loại bài toán, cách sinh cấu hình tiếp theo sẽ thay đổi.

---

3. 🔢 Sinh xâu nhị phân độ dài n

✅ Mô tả:

Sinh tất cả các xâu nhị phân độ dài nnn: gồm các dãy chỉ chứa 0 và 1.

🔷 Code:

```
#include <iostream>
#include <vector>
using namespace std;

// Khởi tạo cấu hình đầu tiên: tất cả các bit đều bằng 0
void init(vector<int>& a) {
   for (int i = 0; i < a.size(); i++) {
      a[i] = 0;
   }
}

// Sinh cấu hình kế tiếp của xâu nhị phân
```

```
void sinh(vector<int>& a, bool& final) {
    int i = a.size() - 1;
    // Duyệt ngược từ cuối về đầu, tìm chữ số 0 đầu tiên để chuyển thành 1
    while (i >= 0 && a[i] == 1) {
        a[i] = 0;
        i--;
    }
    // Nếu đã duyệt hết mà không còn chữ số 0 nào → đã là cấu hình cuối cùng
    if (i == -1) {
        final = true;
    } else {
        a[i] = 1;  // Tìm được chữ số 0 đầu tiên → chuyển thành 1
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    init(a);  // Buoc 1: Khởi tạo cấu hình đầu tiên: toàn 0
    bool final = false;  // Biến đánh dấu kết thúc sinh cấu hình
    // Buoc 2: Vòng lặp sinh tất cả các xâu nhị phân
    while (!final) {
        // + Dua ra cấu hình hiện tại
        for (int x : a) {
            cout << x;
        }
        cout << endl;
        // Sinh cấu hình kế tiếp
        sinh(a, final);
    }
}
```

🔶 Ví dụ (n = 3):

```
000
001
010
011
100
101
110
111
```

---

4. 🧱 Sinh tổ hợp chập K của N phần tử

✅ Mô tả:

Liệt kê tất cả các tổ hợp chập K của tập {1, 2, ..., N}. Mỗi tổ hợp là dãy tăng dần.

🔷 Code:

```
#include <iostream>
#include <vector>
using namespace std;

// Khởi tạo cấu hình đầu tiên từ [1, 2, ..., k]
void init(vector<int>& a) {
    for (int i = 0; i < a.size(); i++) {
        a[i] = i + 1;
    }
}

// Hàm sinh cấu hình kế tiếp của tổ hợp chập k của n phần tử
void sinh(vector<int>& a, bool& final, int n) {
    int i = a.size() - 1;
    int k = a.size();

    // Tìm vị trí đầu tiên mà a[i] chưa đạt giá trị lớn nhất có thể tại vị trí đó
    while (i >= 0 && a[i] == n - k + i + 1) {
        i--;
    }

    // Nếu không còn vị trí nào để tăng → đã sinh hết
    if (i == -1) {
        final = true;
    } else {
        a[i] = a[i] + 1;
        // Cập nhật lại các phần tử phía sau để đảm bảo tính tăng dần
        for (int j = i + 1; j < k; j++){
            a[j] = a[j - 1] + 1;
        }
    }
}

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> a(k);
    init(a);  // Buoc 1: Khởi tạo cấu hình đầu tiên từ [1, 2, ..., k]

    bool final = false;  // Biến đánh dấu kết thúc sinh cấu hình

    // Buoc 2: Vòng lặp sinh tất cả các tổ hợp chập k của n
    while (!final) {
        // + Dua ra cấu hình hiện tại
        for (int x : a) {
            cout << x;
        }
        cout << endl;
```

```
        // Sinh cấu hình kế tiếp
        sinh(a, final, n);
    }
}
```

🔶 Ví dụ (N = 5, K = 3):

```
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
```

---

5. 🔁 Sinh hoán vị của N phần tử

✅ Mô tả:

Sinh tất cả các hoán vị của tập {1, 2, ..., N}. Mỗi hoán vị là dãy gồm N phần tử không trùng nhau.

🔷 Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Khởi tạo cấu hình đầu tiên từ [1, 2, ..., n]
void init(vector<int>& a) {
    for (int i = 0; i < a.size(); i++) {
        a[i] = i + 1;
    }
}

// Hàm sinh hoán vị kế tiếp của dãy a theo thứ tự từ điển
void sinh(vector<int>& a, bool& final) {
    int i = a.size() - 2;

    // Tìm phần tử a[i] đầu tiên (từ phải sang trái) sao cho a[i] < a[i + 1]
    while (i >= 0 && a[i + 1] < a[i]) {
        i--;
    }

    // Nếu không tìm thấy (i == -1) → cấu hình hiện tại là hoán vị cuối cùng
    if (i == -1) {
```

```
        final = true;
    } else {
        // Tìm phần tử a[j] lớn nhất bên phải a[i] mà a[j] > a[i]
        int j = a.size() - 1;
        int temp = 0;
        while (j >= 0 && a[j] < a[i]){
            j--;
        }
        // Hoán đổi a[i] và a[j]
        swap(a[i], a[j]);
        // Đảo ngược đoạn từ a[i + 1] đến hết dãy để đảm bảo thứ tự tăng
        reverse(a.begin() + i + 1, a.end());
    }
}

int main() {
    int n;
    cin >> n;

    vector<int> a(n);
    init(a);  // Buoc 1: Khởi tạo cấu hình đầu tiên:  [1, 2, ..., n]

    bool final = false;  // Biến đánh dấu kết thúc sinh cấu hình

    // Buoc 2: Vòng lặp sinh tất cả các hoán vị
    while (!final) {
        // + Dua ra cấu hình hiện tại
        for (int x : a) {
            cout << x;
        }
        cout << endl;

        // Sinh cấu hình kế tiếp
        sinh(a, final);
    }
}
```

🔶 Ví dụ (N = 3):

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

---

6. 🍰 Sinh phân hoạch của số nguyên N

✅ Mô tả:

Phân hoạch là cách viết N thành tổng các số nguyên dương không tăng.

🔷 Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// Khởi tạo cấu hình đầu tiên từ [n]
void init(vector<int>& a) {
    a[0] = a.size();
}

// Hàm sinh phân hoạch kế tiếp của số nguyên n
void sinh(vector<int>& a, bool& final, int& cur_size) {
    int i = cur_size -1;
    int sum = 0;

    // Duyệt ngược để tìm phần tử đầu tiên lớn hơn 1
    while (i >= 0 && a[i] == 1) {
        i--;
        sum++;
    }
    // Nếu không còn phần tử nào lớn hơn 1 → đã sinh hết phân hoạch
    if (i == -1) {
        final = true;
    } else {
        sum += 1;
        a[i] -= 1;    // Giảm phần tử tại vị trí i đi 1
        int temp = a[i];
        int q = sum / a[i];  // Số lần temp có thể xuất hiện
        int r = sum % a[i];  // Phần dư còn lại
        // Gán q phần tử bằng temp
        for (int j = 1; j <= q; j++) {
            i++;
            a[i] = temp;
        }
        // Nếu còn dư thì thêm phần tử cuối cùng
        if (r != 0) {
            i++;
            a[i] = r;
        }
        // Cập nhật độ dài phân hoạch mới
        cur_size = i + 1;
    }
}

int main() {
    int n;
    cin >> n;
    int cur_size = 1;
```

```
    vector<int> a(n);
    init(a);  // Buoc 1: Khởi tạo cấu hình đầu tiên:  [n]

    bool final = false;  // Biến đánh dấu kết thúc sinh cấu hình

    // Buoc 2: Vòng lặp sinh tất cả các phân hoạch
    while (!final) {
        // + Dua ra cấu hình hiện tại
        for (int i = 0; i < cur_size; i++){
            cout << a[i] << " ";
        }
        cout << endl;

        // Sinh cấu hình kế tiếp
        sinh(a, final, cur_size);
    }
}
```

🔶 Ví dụ (N = 4):

```
4
3 1
2 2
2 1 1
1 1 1 1
```

---

7. Các hàm hỗ trợ

7.1 Hàm next_permutaion

- Hàm `next_permutation` là công cụ hỗ trợ mạnh trong các bài toán sinh hoán vị. Nó giúp tự động sinh ra **hoán vị kế tiếp** của một dãy số theo thứ tự từ điển mà không cần tự cài đặt thuật toán sinh.

🔷 Cú pháp tổng quát

```
bool next_permutation(BidirectionalIterator first, BidirectionalIterator last, Compare comp);
```

◾ Tham số:

- `first`: iterator trỏ đến phần tử đầu tiên của dãy.
- `last`: iterator trỏ đến **vị trí sau phần tử cuối cùng** của dãy.
- `comp`: hàm so sánh tùy chỉnh (ví dụ: `greater<int>()` để sinh hoán vị theo thứ tự giảm dần). *(mặc định là tăng dần)*

◾ Giá trị trả về:

- `true` nếu tồn tại hoán vị kế tiếp (dãy được thay đổi).
- `false` nếu dãy đang ở hoán vị cuối cùng (sẽ trở về hoán vị đầu tiên — thứ tự tăng dần).

✅ Ví dụ:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main() {
    vector<int> a = {1, 2, 3};
    do {
        for (int x : a) cout << x << " ";
        cout << endl;
    } while (next_permutation(a.begin(), a.end()));
}
```

Kết quả:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

7.2 Hàm prev_permutaion

- Hàm `prev_permutation` là công cụ hỗ trợ mạnh trong các bài toán sinh hoán vị **ngược**. Nó giúp tự động sinh ra **hoán vị trước đó** của một dãy số theo thứ tự từ điển mà không cần tự cài đặt thuật toán sinh ngược.

🔷 Cú pháp tổng quát

```
bool prev_permutation(BidirectionalIterator first, BidirectionalIterator last, Compare comp);
```

◾ Tham số:

- `first`: iterator trỏ đến phần tử đầu tiên của dãy.
- `last`: iterator trỏ đến **vị trí sau phần tử cuối cùng** của dãy.
- `comp`: hàm so sánh tùy chỉnh (ví dụ: `greater<int>()` để sinh hoán vị ngược theo thứ tự tăng dần). *(mặc định là giảm dần)*

◾ Giá trị trả về:

- `true` nếu tồn tại hoán vị trước đó (dãy được thay đổi).
- `false` nếu dãy đang ở hoán vị đầu tiên (sẽ trở về hoán vị cuối cùng — thứ tự giảm dần).

✅ Ví dụ:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
```

```
int main() {
    vector<int> a = {3, 2, 1};
    do {
        for (int x : a) cout << x << " ";
        cout << endl;
    } while (prev_permutation(a.begin(), a.end()));
}
```

Kết quả:

```
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3
```

✅ Tổng kết

Kỹ thuật	Đặc điểm chính	Số lượng cấu hình
Nhị phân	Duyệt tất cả dãy 0/1	2^n
Tổ hợp	Dãy tăng, không trùng	$\binom{n}{k}$
Hoán vị	Dãy không lặp, mọi vị trí khác nhau	n!
Phân hoạch	Tổng bằng N, dãy không tăng	p(n)