CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

SILDE 2.4: ĐỆ QUY & QUAY LUI

Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Khái niệm đệ quy
- Thuật toán quay lui
- Các bài toán ví dụ:
  - Sinh xâu nhị phân
  - Sinh hoán vị
  - Sinh tổ hợp
  - Phân tích số
  - Xếp quân hậu

1. Khái niệm đệ quy

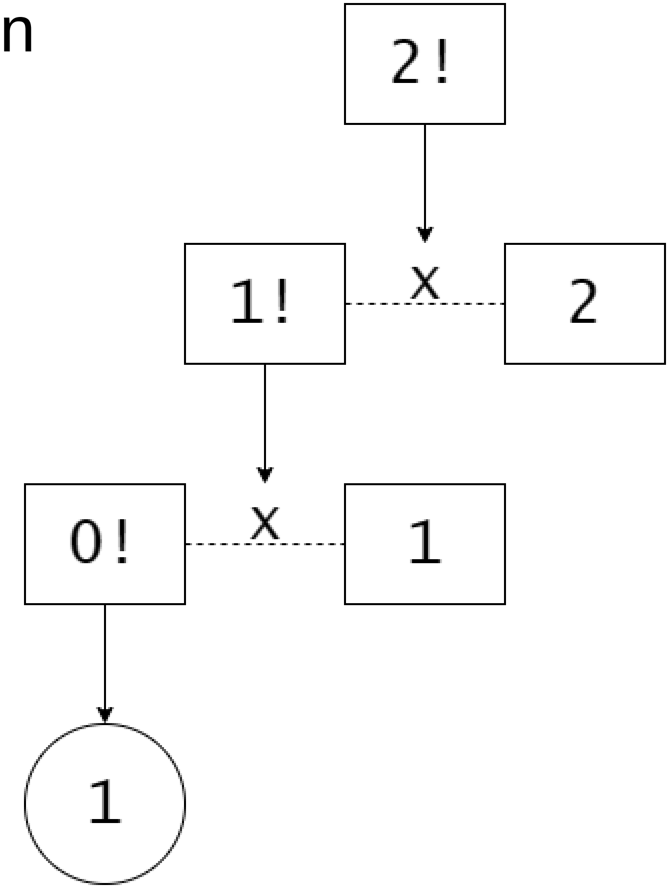
- Ta gọi một đối tượng là đệ quy (recursion) nếu nó được định nghĩa qua chính nó hoặc một đối tượng cùng dạng với chính nó bằng quy nạp.
- Một hàm nếu gọi lại chính nó sẽ được gọi là hàm đệ quy.
- Số lần gọi này phải có giới hạn (điểm dừng).
- Ví dụ:

```
// Đây là một hàm gọi đệ quy
void f() {
    f();
}
```

1. Khái niệm đệ quy

Ví dụ 1: hàm tính giai thừa n!

- Công thức đệ quy: n! = (n-1)! * n
- Hay là f(n) = f(n-1)*n



1. Khái niệm đệ quy

- Ví dụ 1: hàm tính giai thừa

```
void factorial(int n) {
    if (n == 0) return 1;              //trường hợp cơ sở
    return factorial(n - 1) * n;       //phần đệ quy
}
```

Tương đương với

```
//n = 0
void factorial_0() {
    return 1;
}

//n = 1
void factorial_1() {
    return factorial_0() * 1;
}

//n = 2
void factorial_2() {
    return factorial_1() * 2;
}
```

1. Khái niệm đệ quy

- Ví dụ 2: Dãy số Fibonacci

$$f_n = \begin{cases} 0 & \text{với } n = 0 \\ 1 & \text{với } n = 1 \\ f_{n-2} + f_{n-1} & \text{với } n > 1 \end{cases}$$

```
void fibo(int n) {
    if (n == 0) return 0;                //trường hợp cơ sở
    if (n == 1) return 1;                //trường hợp cơ sở
    return fibo(n - 2) + fibo(n - 1);    //phần đệ quy
}
```

1. Khái niệm đệ quy

- Hàm đệ quy phải có 2 phần:
  - Phần dừng: trường hợp suy biến. Trong ví dụ trong bài toán tính giai thừa thì với n=0 là dừng lại.
  - Phần đệ quy: là phần có gọi lại hàm đang được định nghĩa. Trong ví dụ trên thì phần đệ quy là n>0 thì n! = n * (n-1)!
- Sử dụng hàm đệ quy trong chương trình sẽ làm chương trình dễ đọc, dễ hiểu và vấn đề được nêu bật rõ ràng hơn. Tuy nhiên trong đa số trường hợp thì hàm đệ quy tốn bộ nhớ nhiều hơn và tốc độ thực hiện chương trình chậm hơn không đệ quy.
- Tùy bài toán cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không (có những trường hợp không dùng đệ quy thì không giải quyết được bài toán).

1. Khái niệm đệ quy

- Cấu trúc tổng quát

```
If (suy biến) {
        <Giải quyết trường hợp suy biến>;
}
Else {
        <Tiền xử lý đệ qui>;
        <Gọi đệ qui>;
        <Xử lý hậu đệ qui>;
}
```

1. Khái niệm đệ quy

- Ưu điểm:
  - Sáng sủa, dễ hiểu, nêu rõ bản chất vấn đề
  - Tiết kiệm thời gian hiện thực mã nguồn
- Nhược điểm:
  - Tốn nhiều bộ nhớ, thời gian thực thi lâu
  - Một số bài toán không có lời giải đệ quy

2. Thuật toán quay lui

- Quay lui (backtracking) là một kĩ thuật thiết kế giải thuật dựa trên **đệ quy**, dùng để giải bài toán liệt kê các cấu hình. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn có thể và gọi đệ quy cho bước tiếp theo.
- Nói cách khác, chúng ta đang xây dựng một danh sách gồm tất cả các tập hợp (hay dãy, ...), mà mỗi phần tử được xét tất cả các trường hợp có thể của nó. Phương pháp này cũng gọi là **duyệt vét cạn**.

2. Thuật toán quay lui

- Bài toán:
  - Cần xác định bộ X =(x_1, x_2,..,x_k) thỏa mãn ràng buộc.
  - Mỗi thành phần x_i ta có n_i khả năng cần lựa chọn.

```
void backtrack(int pos){
    // Trường hợp cơ sở
    if (<pos là vị trí cuối cùng>){
        <output/lưu lại tập hợp đã dựng nếu thoả mãn>
        return;
    }

    //Phần đệ quy
    for (<tất cả giá trị i có thể ở vị trí pos>){
        <thêm giá trị i vào tập đanh xét>
        backtrack(pos + 1);
        <xoá bỏ giá trị i khỏi tập đang xét>
    }
}
```

3.1 Bài toán ví dụ

(1) Sinh xâu nhị phân có N kí tự (N <= 20)

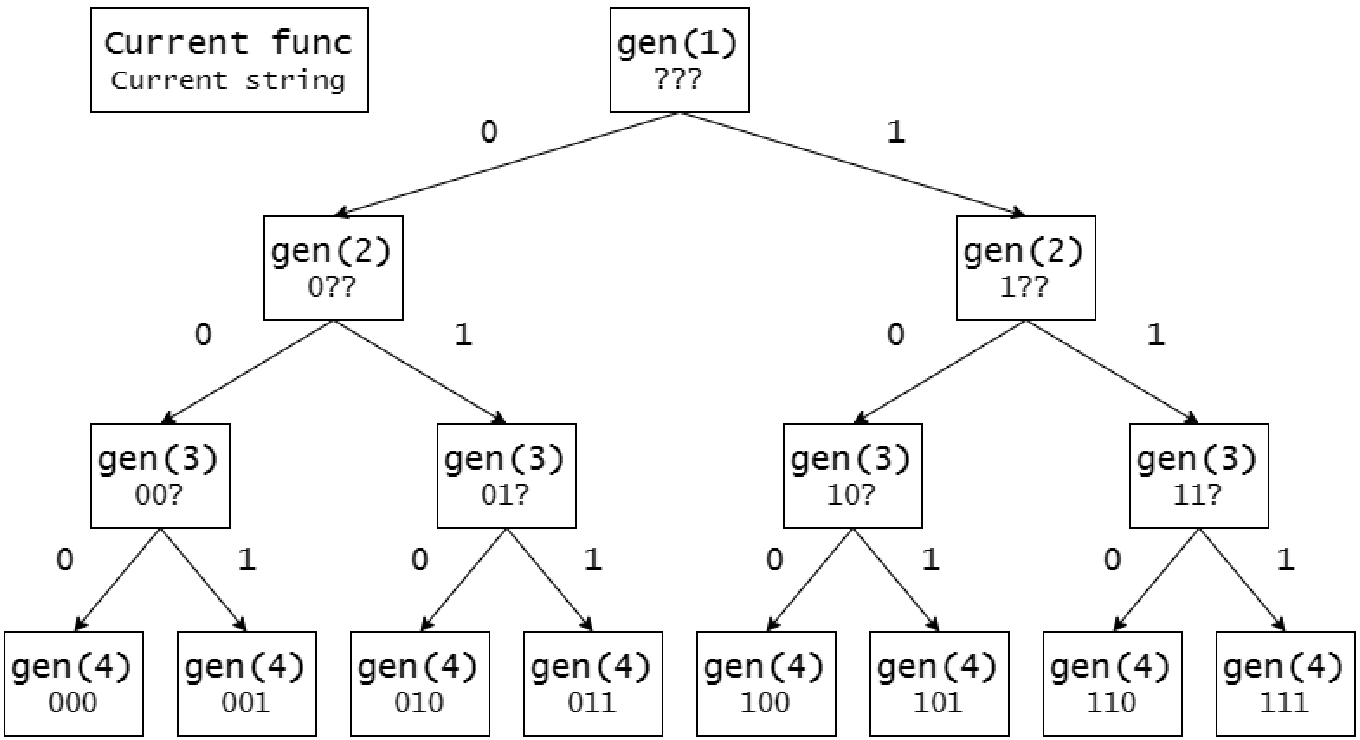
- Với N = 3, có tất cả 8 xâu:
  - 000
  - 001
  - 010
  - 011
  - 100
  - 101
  - 110
  - 111

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

int n;
char ans[21];

void backtrack(int pos){
    if (pos > n){
        for(int i = 1; i <= n; i++)
            cout << ans[i];
        cout << endl;
        return;
    }
    for (char i = '0'; i <= '1'; i ++){
        ans[pos] = i;
        backtrack(pos + 1);
    }
}

int main(){
    cin >> n;
    backtrack(1);

    return 0;
}
```

3.1 Bài toán ví dụ

(1) Sinh xâu nhị phân có N kí tự (N <= 20)

- Với N = 3, có tất cả 8 xâu:



3.2 Bài toán ví dụ

(2) Sinh hoán vị

Cho S = {1, 2, …, N}.

Hãy sinh ra hoán vị của tập S.

Với N = 3, có 6 hoán vị:

- (1, 2, 3)
- (1, 3, 2)
- (2, 1, 3)
- (2, 3, 1)
- (3, 1, 2)
- (3, 2, 1)

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

int n;
int a[21], visited[21];

void backtrack(int pos){
    if (pos > n){
        for(int i = 1; i <= n; i++)
            cout << a[i] << " ";
        cout << endl;
        return;
    }
    for (int i = 1; i <= n; i++){
        if(!visited[i]) {
            a[pos] = i;
            visited[i] = 1;
            backtrack(pos + 1);
            visited[i] = 0;
        }
    }
}

int main(){
    memset(visited, 0, sizeof(visited));
    cin >> n;
    backtrack(1);

    return 0;
}
```

3.3 Bài toán ví dụ

(3) Sinh tổ hợp

Cho S = {1, 2, …, N}. Hãy sinh ra các cấu hình có K phần tử riêng biệt.

Với N = 4, K = 2, có 6 tổ hợp:

- (1, 2)
- (1, 3)
- (1, 4)
- (2, 3)
- (2, 4)
- (3, 4)

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

int n, k;
int a[21];

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

int main(){
    cin >> n >> k;
    a[0] = 0;
    backtrack(1);

    return 0;
}
```

3.4 Bài toán ví dụ

(4) Bài toán phân tích số

- Cho số tự nhiên N (N≤100) và tập X = {x_1, x_2, …, x_k}. Hãy liệt kê các cách phân tích N thành tổng các phần tử trong tập X.

(Không tính các hoán vị)

- Ví dụ với N = 100, X = {10, 20, 50}, ta có 10 cách chia như sau:
  - 10 10 10 10 10 10 10 10 10 10
  - 50 50
  - 10 20 20 50
  - ….

3.4 Bài toán ví dụ

(4) Bài toán phân tích số

- Ví dụ với N = 100, X = {10, 20, 50},

Ta có 10 cách chia như sau:

- 10 10 10 10 10 10 10 10 10 10
- 10 10 10 10 10 10 10 10 20
- 10 10 10 10 10 10 20 20
- 10 10 10 10 10 50
- 10 10 10 10 20 20 20
- 10 10 10 20 50
- 10 10 20 20 20 20
- 10 20 20 50
- 20 20 20 20 20
- 50 50

```
int n = 100;
int a[50], X[4] = {0, 10, 20, 50};
int sum = 0, cnt = 0;

void backtrack(int pos){
    if (sum >= n){
        if (sum == n) {
            for(int i = 1; i <= pos-1; i++)
                cout << a[i] << " ";
            cout << endl;
        }
        return;
    }
    for (int i = 1; i <= 3; i++){
        a[pos] = X[i];
        sum += X[i];
        backtrack(pos + 1);
        sum -= X[i];
    }
}

int main(){
    backtrack(1);
    return 0;
}
```

Lưu ý: Đoạn code trên sinh ra các cấu hình bị lặp, các bạn cần sửa một ít

3.5 Bài toán ví dụ - Kĩ thuật nhánh cận

(5) Phân tích số: in ra 1 kết quả có ít phần tử nhất

Cho số tự nhiên N (N≤100) và tập X = $\{x_1, x_2, \ldots, x_k\}$. Hãy liệt kê các cách phân tích N thành tổng các phần tử trong tập X.

Yêu cầu in ra cách phân tích có **ít phần tử nhất** có thể.

- Ví dụ với N = 100, X = {10, 20, 50}, ta có 10 cách chia như sau:
  - 10 10 10 10 10 10 10 10 10 10
  - 10 20 20 50
  - …..
  - **50 50**

3.5 Bài toán ví dụ - Kĩ thuật nhánh cận

(5) Phân tích số: in ra 1 kết quả có ít phần tử nhất

- Kỹ thuật nhánh cận (branch and bound) dùng để giải quyết các bài toán in ra 1 phương án tối ưu, thay vì yêu cầu liệt kê ra tất cả các cấu hình thỏa mãn.
- **Đặc điểm:** nếu như tại 1 bước nào đó, cấu hình đang duyệt không thể sinh ra một kết quả tốt hơn cấu hình cũ nào đó, chúng ta có thể bỏ qua nó luôn.

3.5 Bài toán ví dụ - Kĩ thuật nhánh cận

(5) Phân tích số: in ra 1 kết quả có ít phần tử nhất

- Ví dụ với N = 100, X = {10, 20, 50}, quá trình duyệt sinh ra 10 cấu hình, đáp án tối ưu là cấu hình sinh ra cuối cùng:
  - 10 10 10 10 10 10 10 10 10 10
  - 10 20 20 50
  - …..
  - **50 50**

- Sau khi thêm điều kiện nhánh cận:

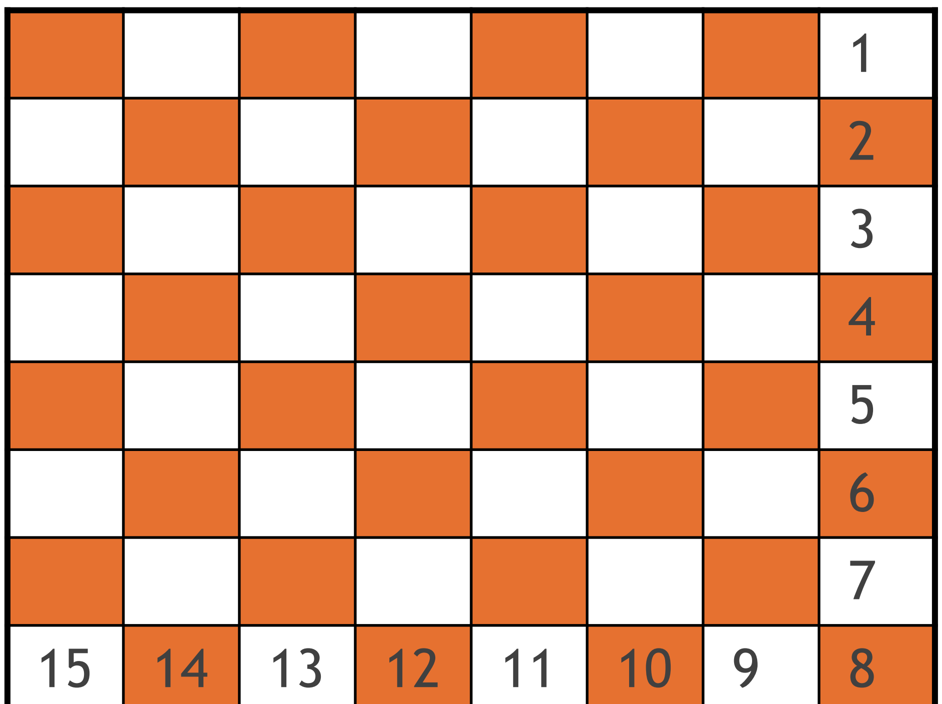
Với N = 100, nhưng X = {50, 20, 10}, quá trình duyệt chỉ sinh ra một cấu hình duy nhất (cấu hình đầu tiên):

  - **50 50**

6. Bài toán ví dụ

(6) Bài toán xếp quân hậu

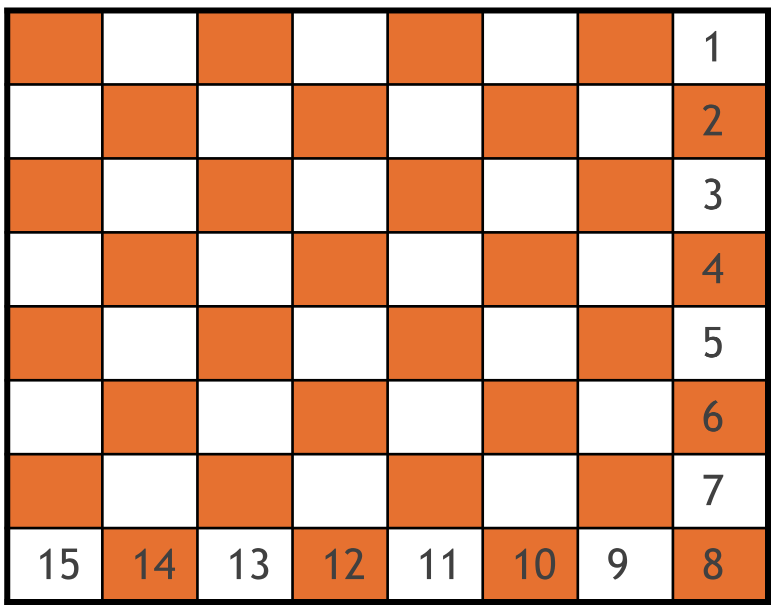
- Cho bàn cờ 8x8, hãy xếp 8 quân hậu trên bàn cờ sao không có 2 quân hậu nào xung đột với nhau.



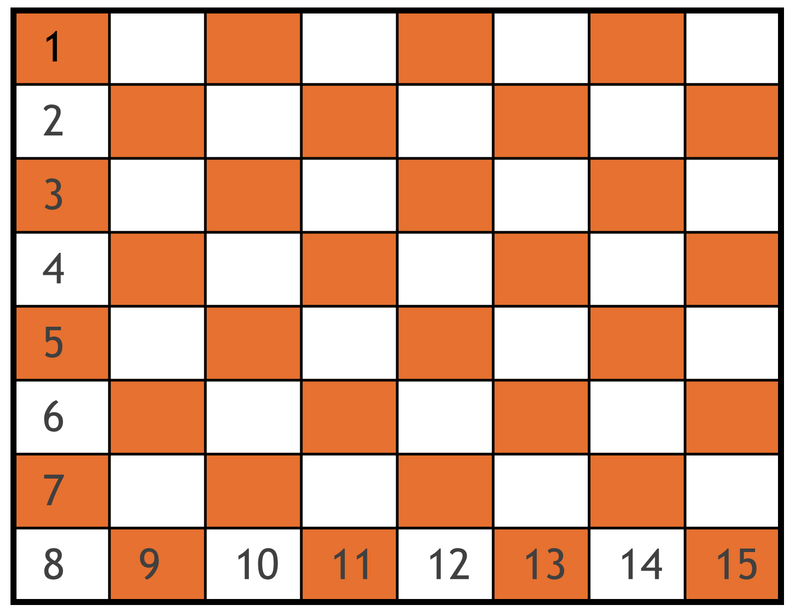
6. Bài toán ví dụ

(6) Bài toán xếp quân hậu

- Cho bàn cờ 8x8, hãy xếp 8 quân hậu trên bàn cờ sao không có 2 quân hậu nào xung đột với nhau.



Đường chéo xuôi: Xuoi [i – j + n]



Đường chéo ngược: Nguoc [i + j -1]

6. Bài toán ví dụ

(6) Bài toán xếp quân hậu

- Cho bàn cờ 8x8, hãy xếp 8 quân hậu trên bàn cờ sao không có 2 quân hậu nào xung đột với nhau.
- Điều kiện:
  - (1) Không có 2 quân hậu nào cùng 1 hàng
  - (2) Không có 2 quận hậu nào cùng 1 cột
  - (3) Không có 2 quân hậu cùng nằm trên đường chéo chính
  - (4) Không có 2 quân hậu cùng nằm trên đường chéo phụ (ngược)

QUESTIONS & ANSWERS