CHƯƠNG 3
ĐỆ QUY VÀ CHIẾN LƯỢC VÉT CẠN

Nội dung chính chương 3

3.1 Khái niệm đệ quy

3.2 Chiến lược vét cạn (Brute force)

3.3 Chiến lược quay lui (Backtracking/try and error)

3.4 Kỹ thuật băm

3.1 Khái niệm đệ quy

- ✓ Đệ quy là cách định nghĩa một đối tượng dựa trên chính nó.
- ✓ Trong toán học hay sử dụng phương pháp chứng minh quy nạp, chính là một nguyên lý đệ quy.
- ✓ Trong lập trình một hàm đệ quy là một hàm mà trong thân hàm (cài đặt) có lời gọi tới chính nó (số lượng và vị trí không hạn chế).
- ✓ Giải thuật đệ quy là giải thuật dựa trên các quan hệ đệ quy và được cài đặt cụ thể bằng các hàm đệ quy.

3.1 Khái niệm đệ quy

Ví dụ:

$$\begin{cases} 0!=1 \\ n!= n*(n-1)! \quad n>0 \end{cases}$$

Hàm Factorial

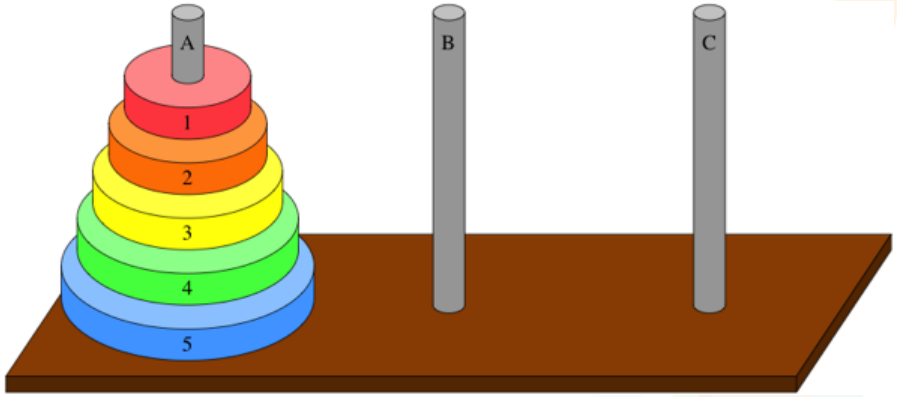
Factorial(0)=1

Factorial(n)=n*Factorial(n-1)

3.1. Khái niệm đệ quy

Bài toán tháp Hà Nội:

Bài toàn tháp Hà Nội là trò chơi toán học gồm 3 cọc và n đĩa có kích thước khác nhau. Ban đầu các đĩa được xếp chồng lên nhau trong cọc A như hình vẽ. Yêu cầu của bài toán:

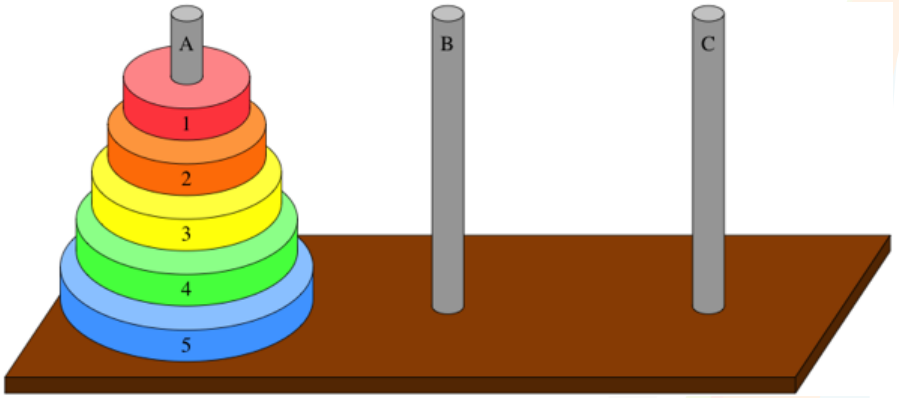


- Di chuyển toàn bộ các đĩa ở cọc A sang cọc C với điều kiện sau.
- Mỗi lần thực hiện chỉ được di chuyển một đĩa
- Các đĩa phải xếp theo nguyên tắc, đĩa lớn ở dưới, đĩa nhỏ ở trên.
- Được phép thêm một cọc B làm trung gian để di chuyển các đĩa.

3.1. Khái niệm đệ quy

Bài toán tháp Hà Nội:

Với tư duy quy nạp toán học và sức mạnh của đệ quy, bài toán này có thể được giải quyết dễ dàng như sau:



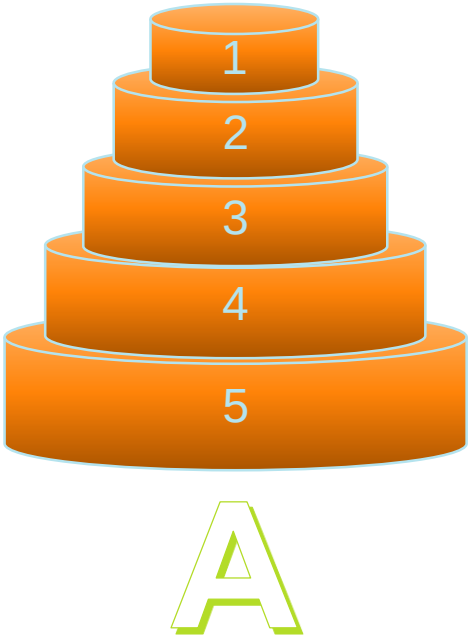
- Xét N đĩa ban đầu đặt ở cọc A. Nếu N=1 thì chuyển đĩa đó từ cọc A sang cọc C là xong.
- Ta lấy cọc B làm cọc trung gian. Bài toán được phân tách ra làm hai phần: Chuyển $N-1$ đĩa từ cọc A sang cọc trung gian B, rồi chuyển đĩa ở cuối cùng sang cọc C, cuối cùng chuyển lại $N-1$ chiếc đĩa ở cọc B sang cọc C bằng cách lấy cọc A làm trung gian.

3.1. Khái niệm đệ quy

```
void tower(int N, int a, int b, int c)
{
    if(N == 1)
    {
        cout << a << " ---> " << c <<endl;
        return;
    }
    tower(N - 1, a, c, b);
    tower(1, a, b, c);
    tower(N - 1, b, a, c);
}
int main()
{
    int N;
    cin >> N;
    int a = 1, b = 2, c = 3;
    tower(N, a, b, c);
}
```

3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội

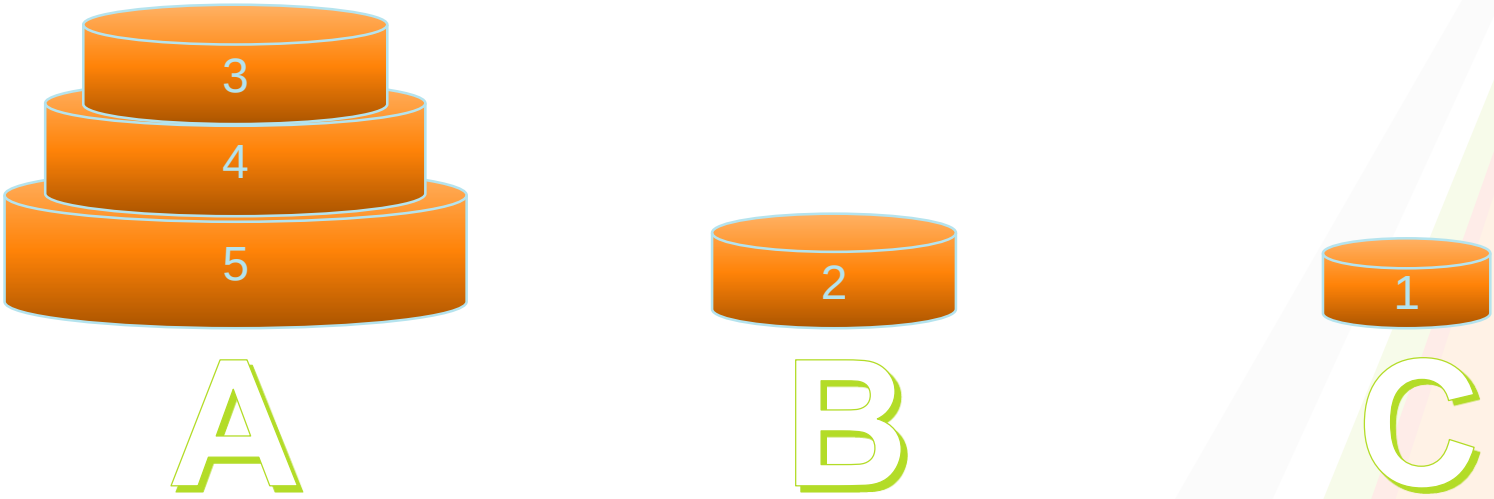


3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội

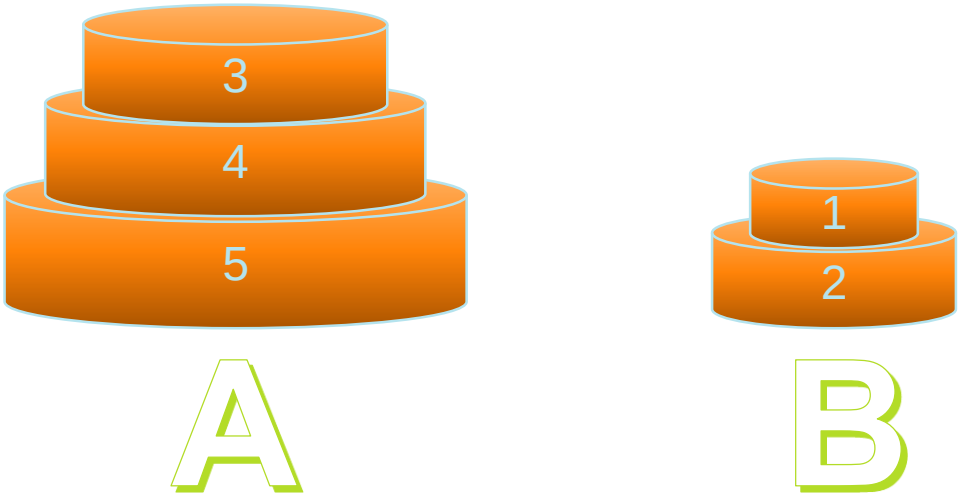
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



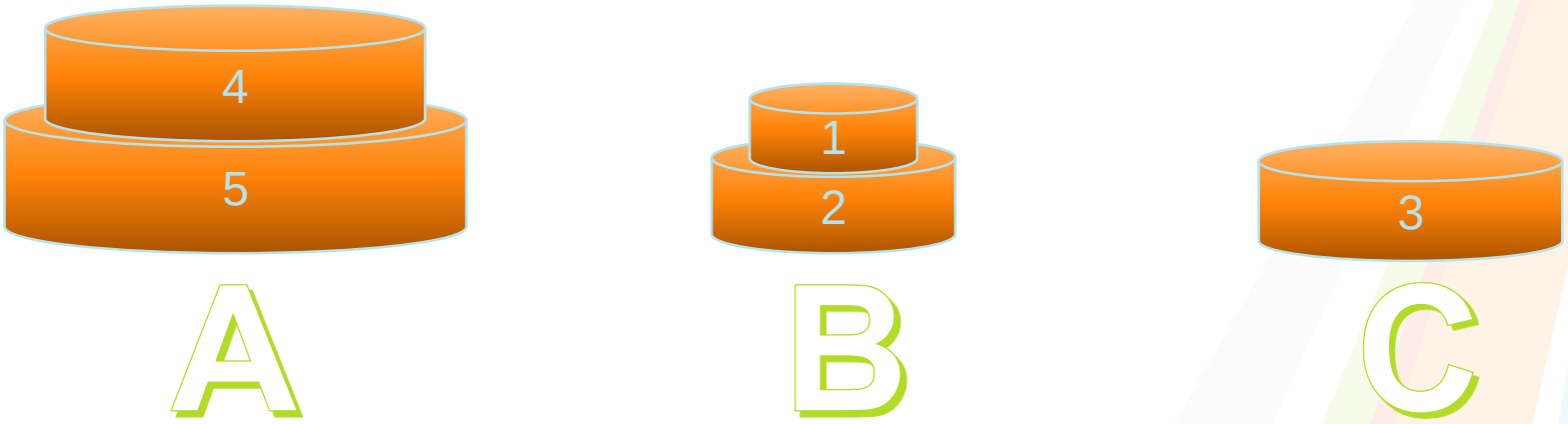
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



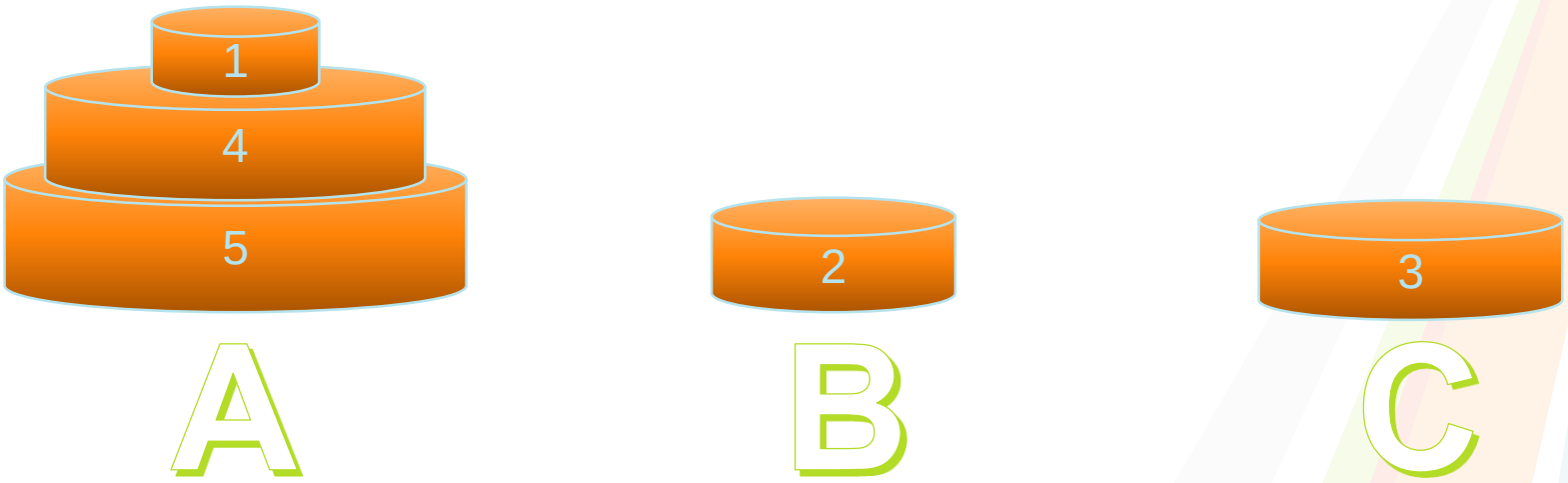
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



3.1. Khái niệm đệ quy

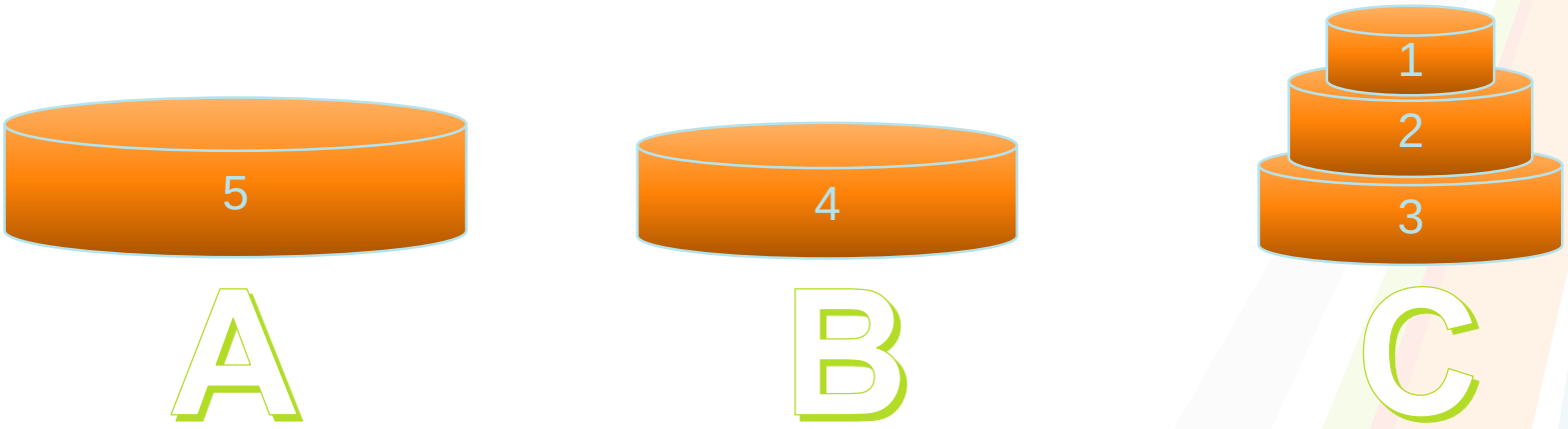
Minh họa bài toán tháp Hà Nội

3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội

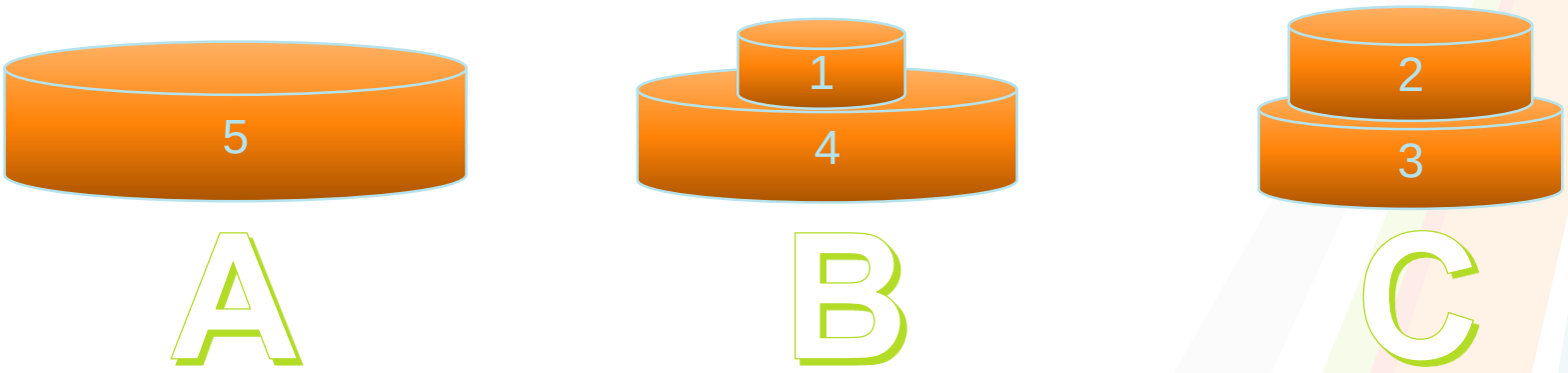
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



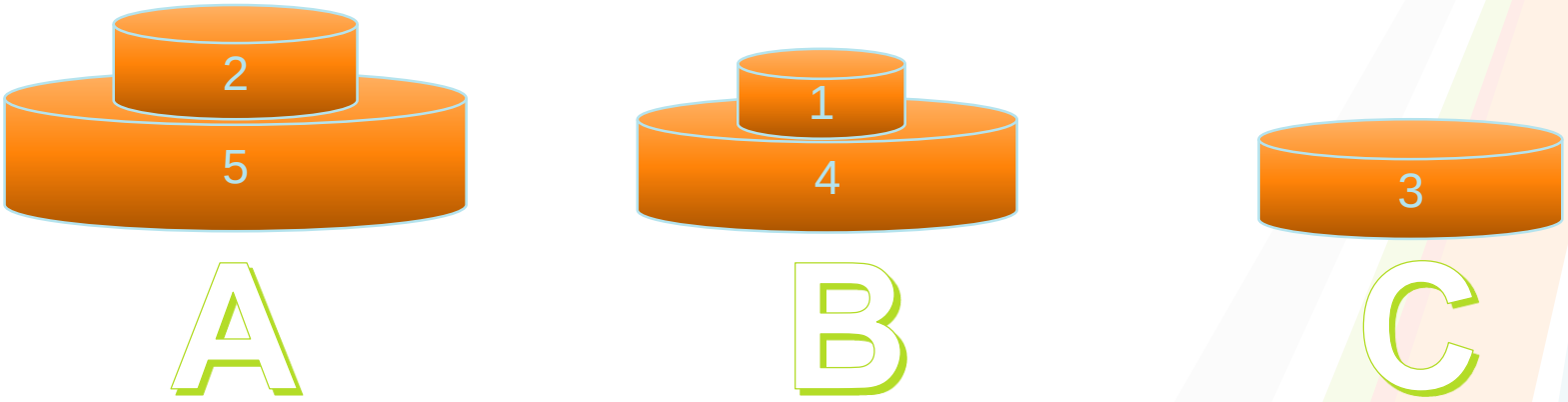
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



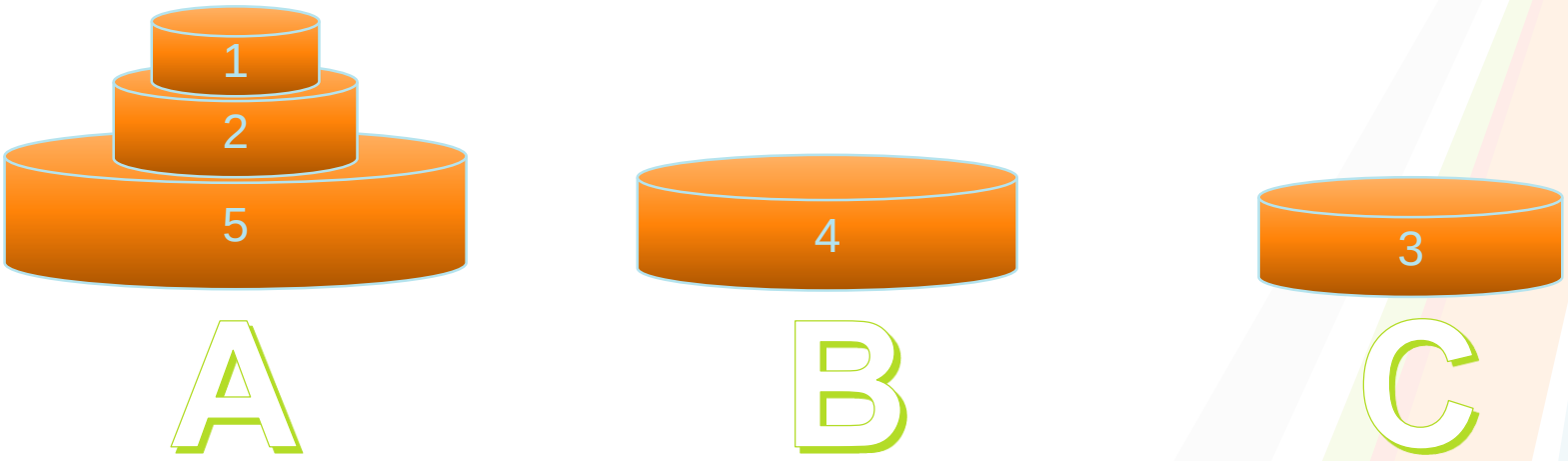
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



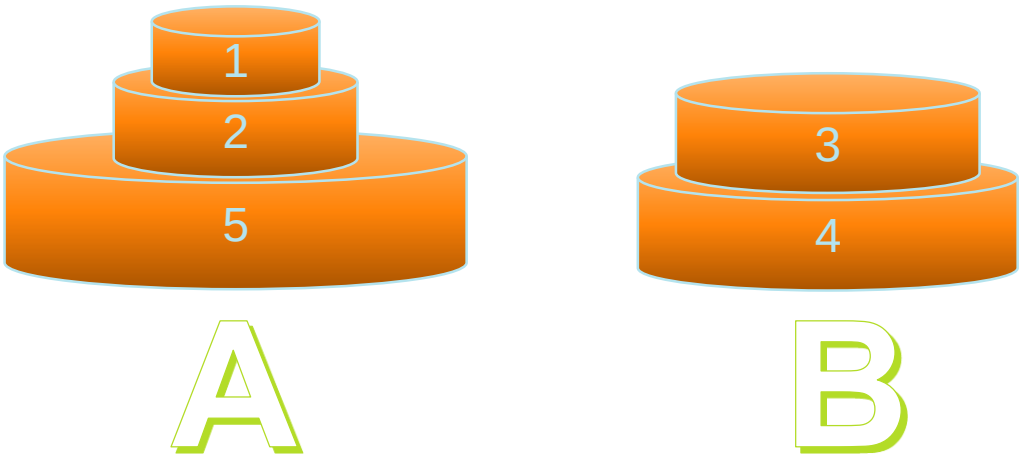
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



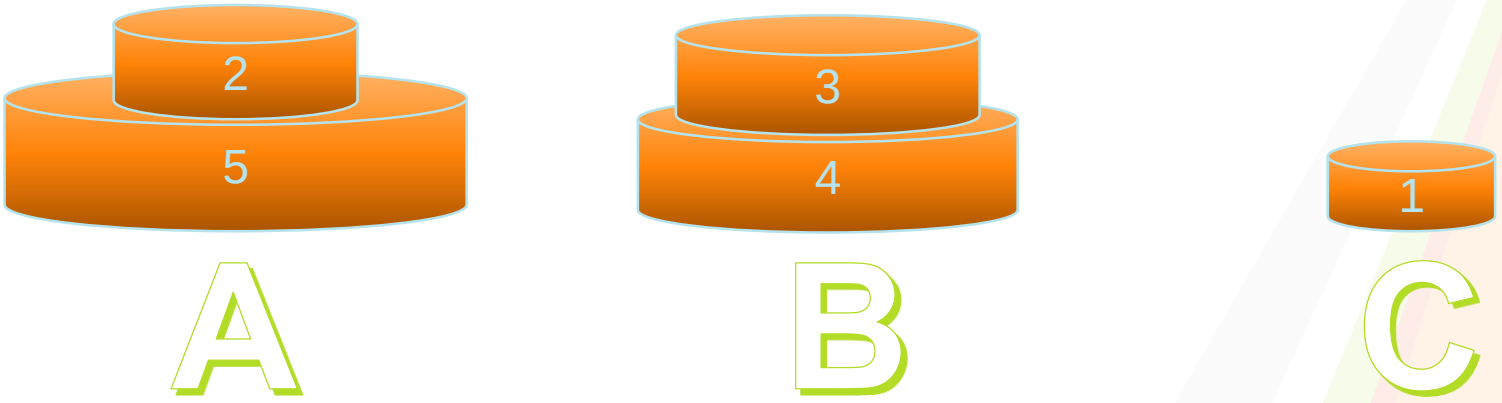
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



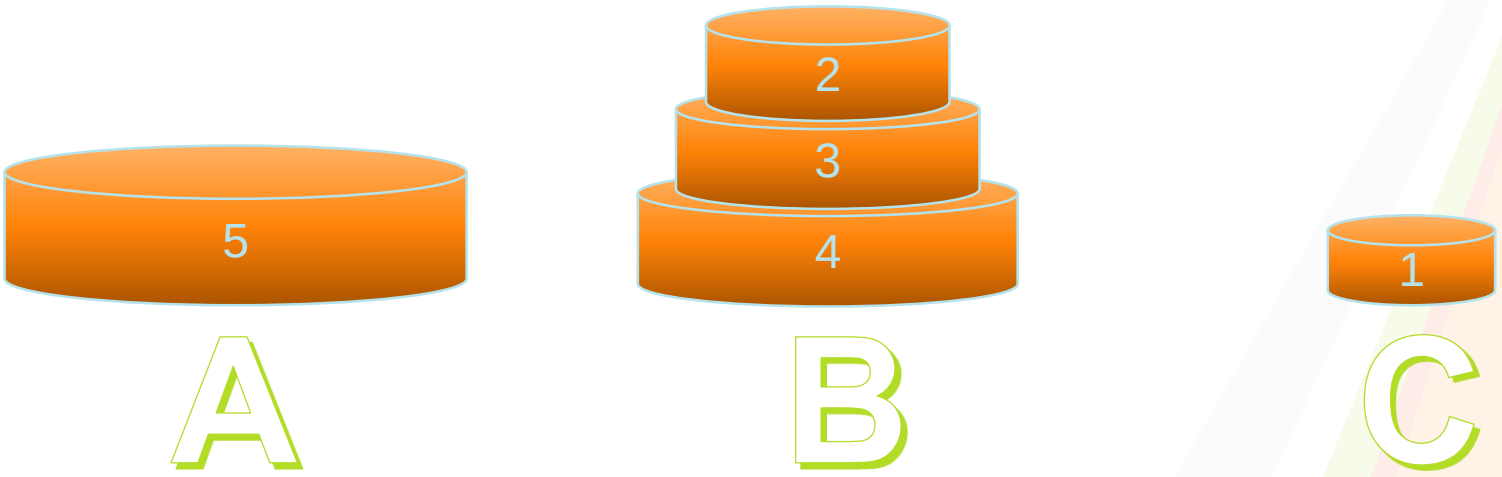
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



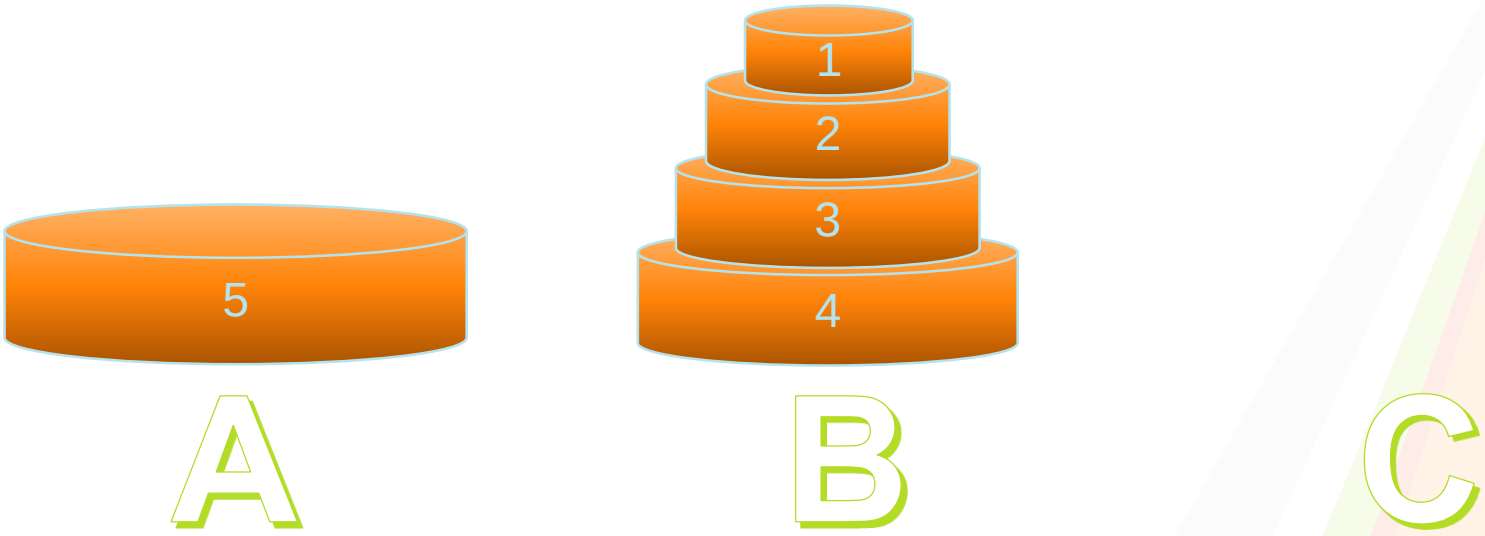
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



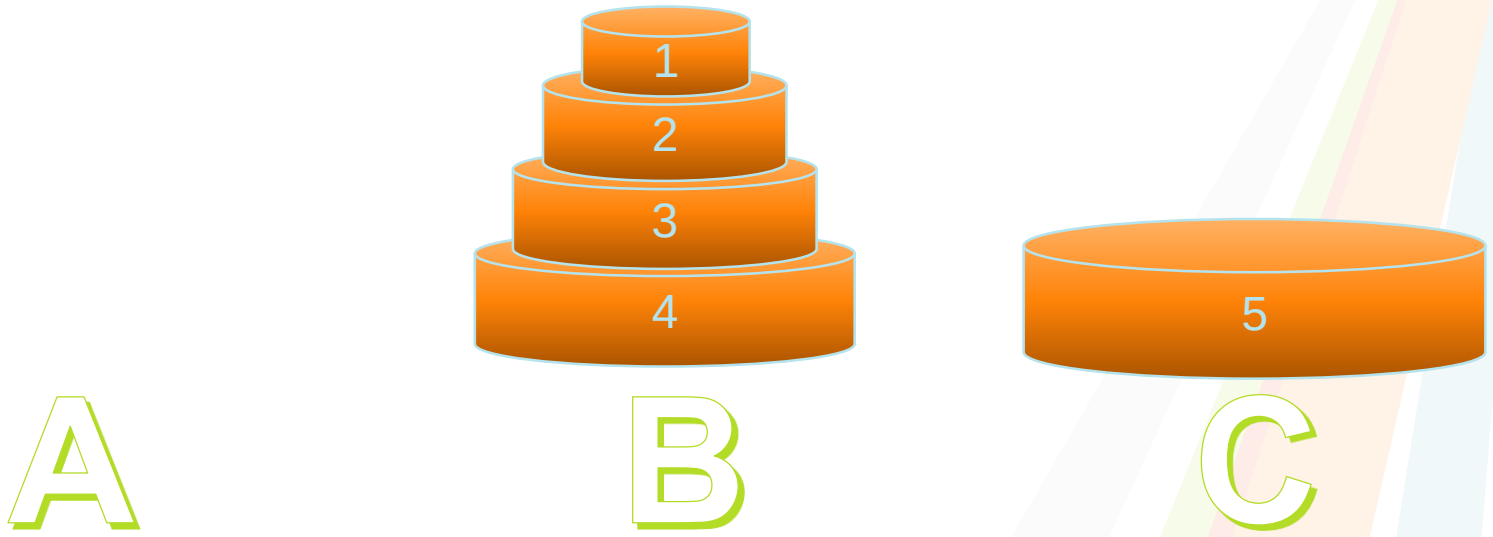
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



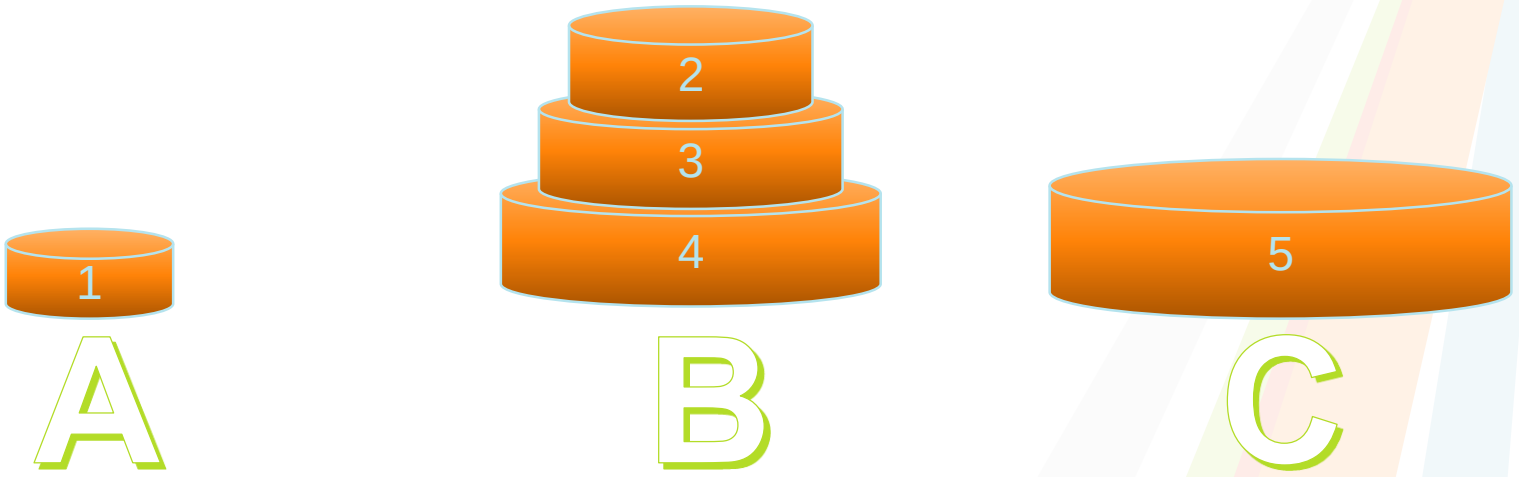
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



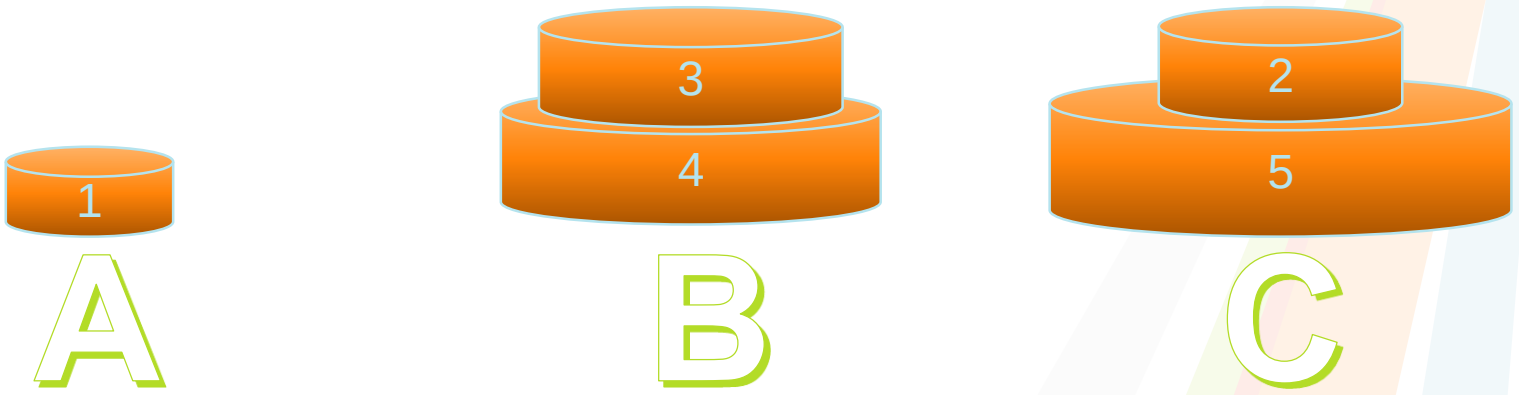
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



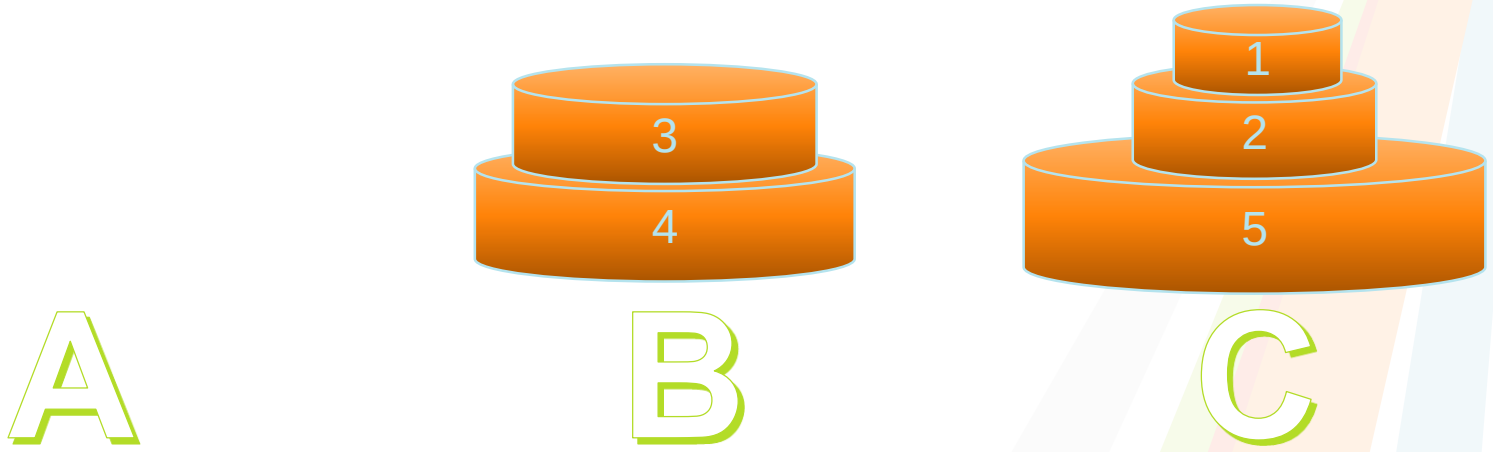
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



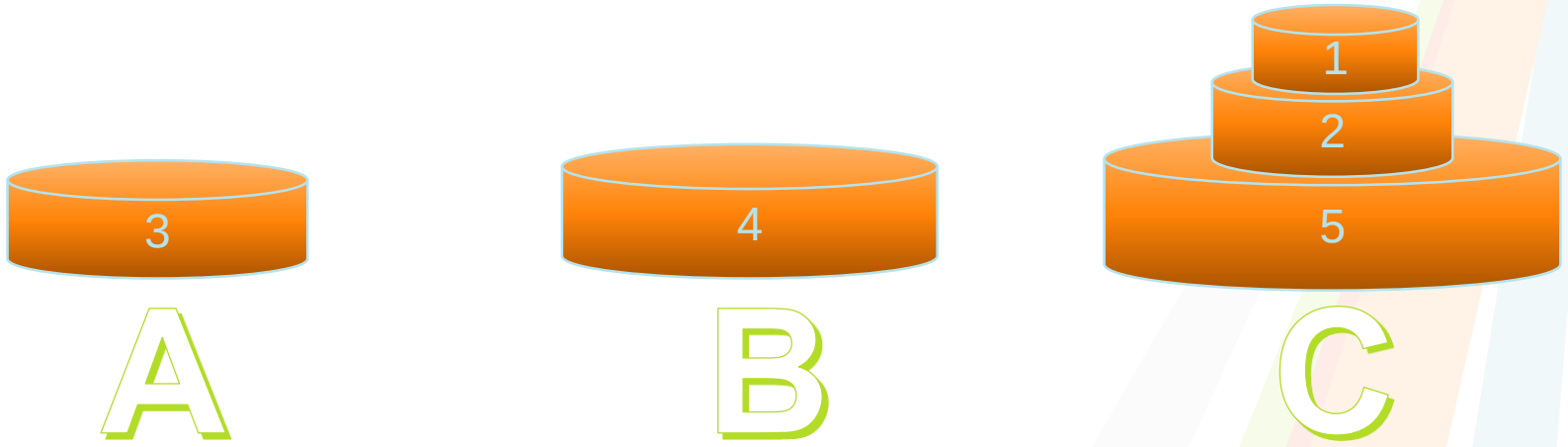
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



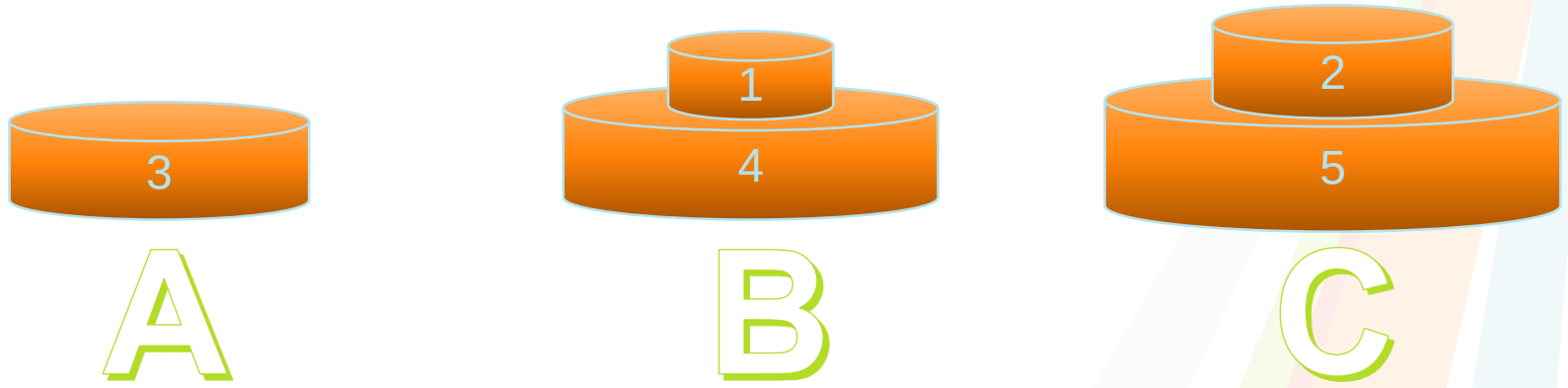
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



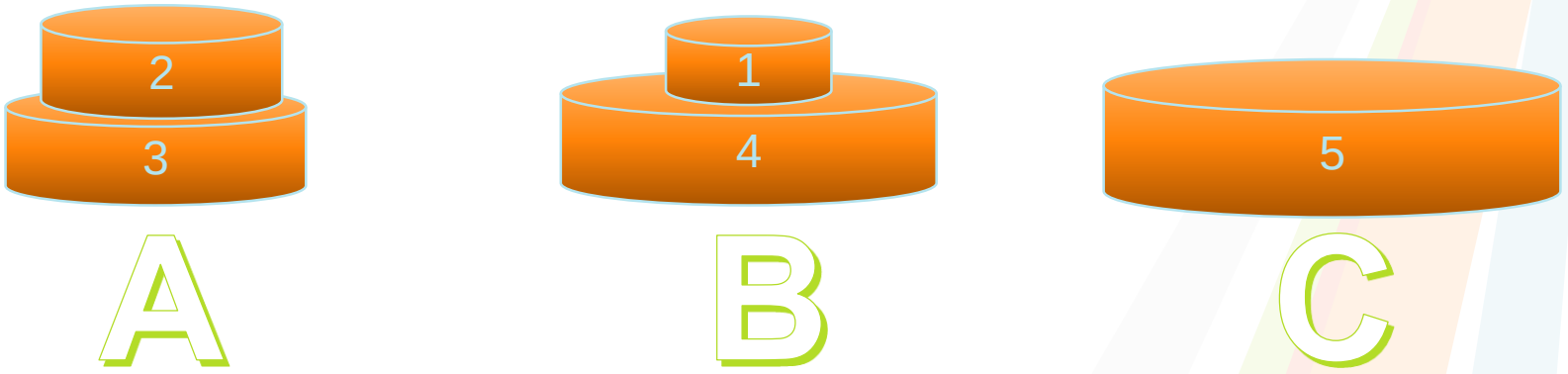
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



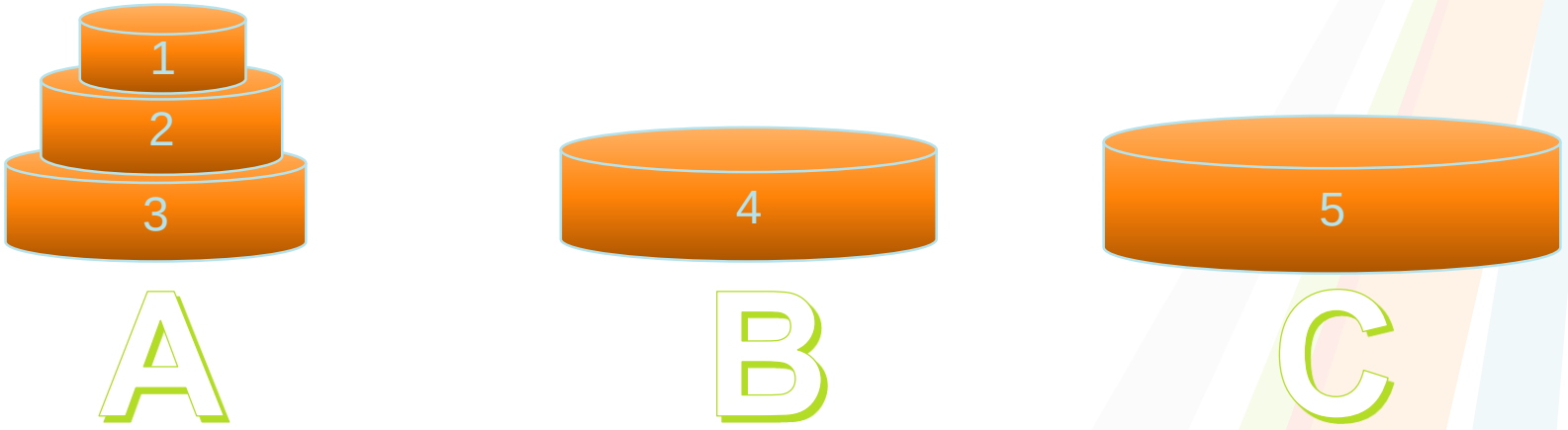
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



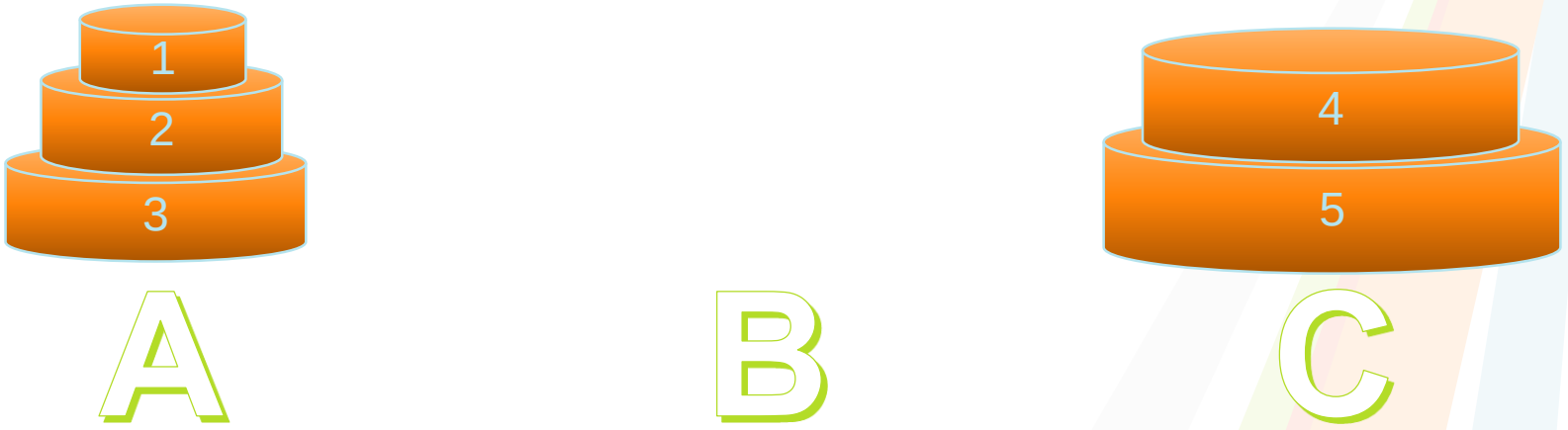
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



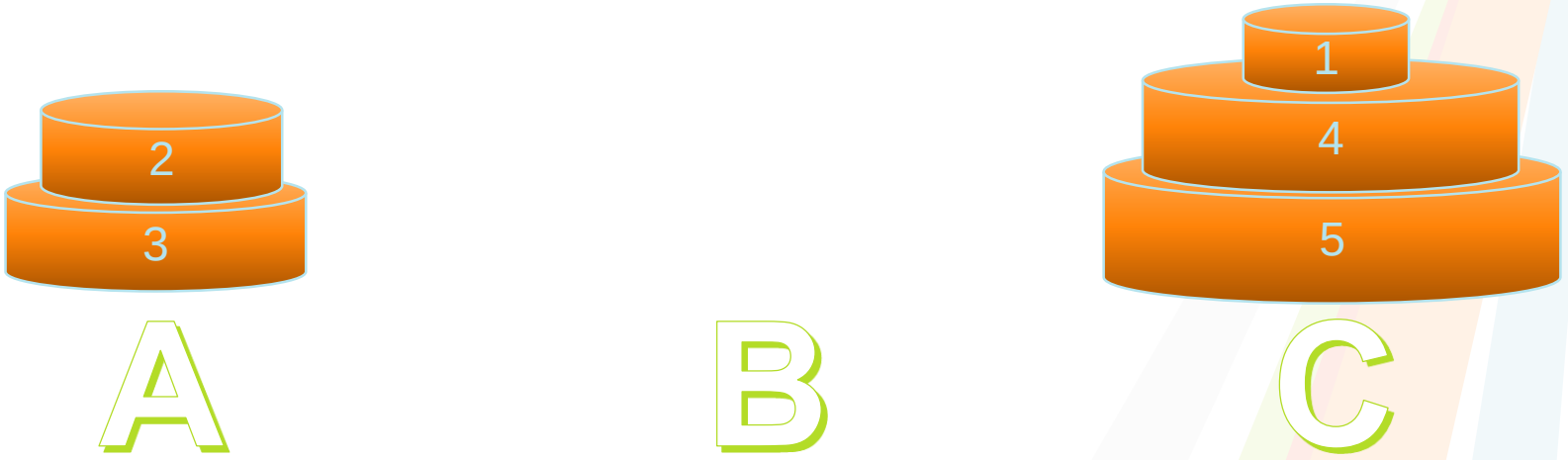
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



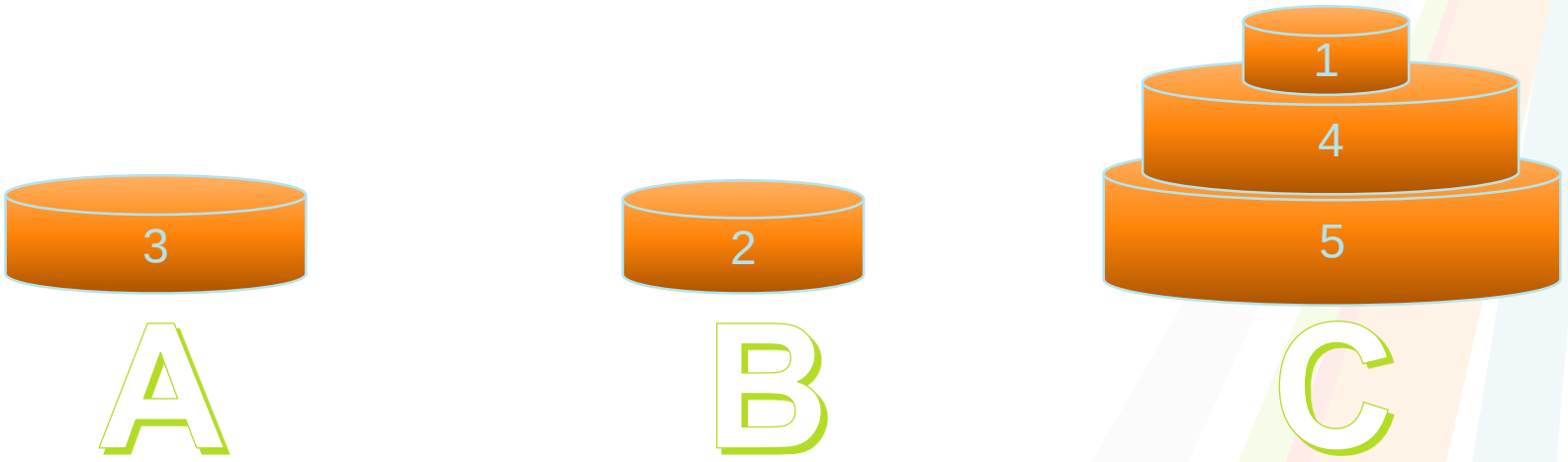
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



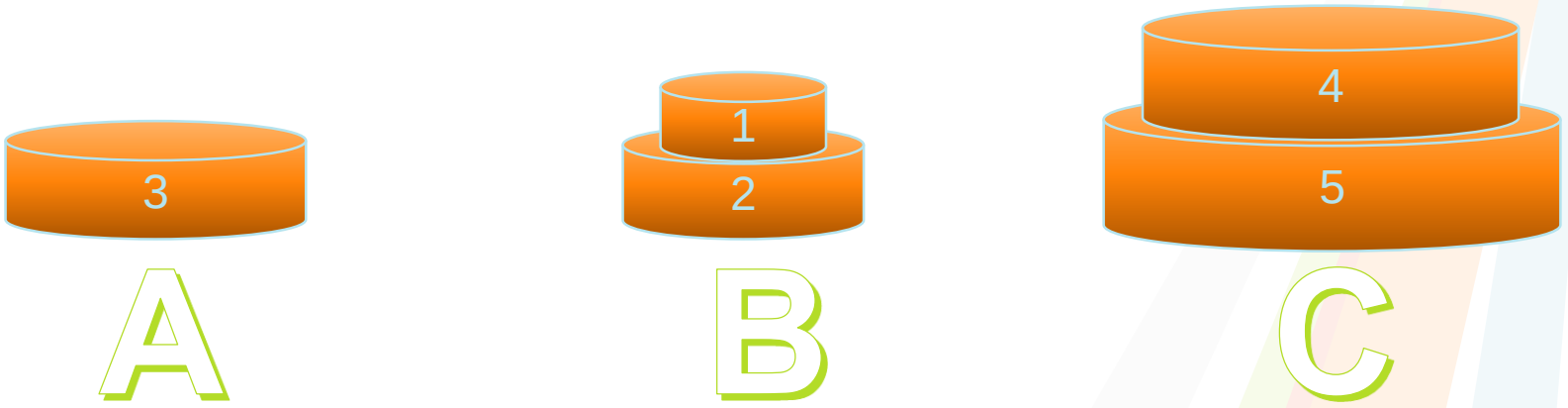
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



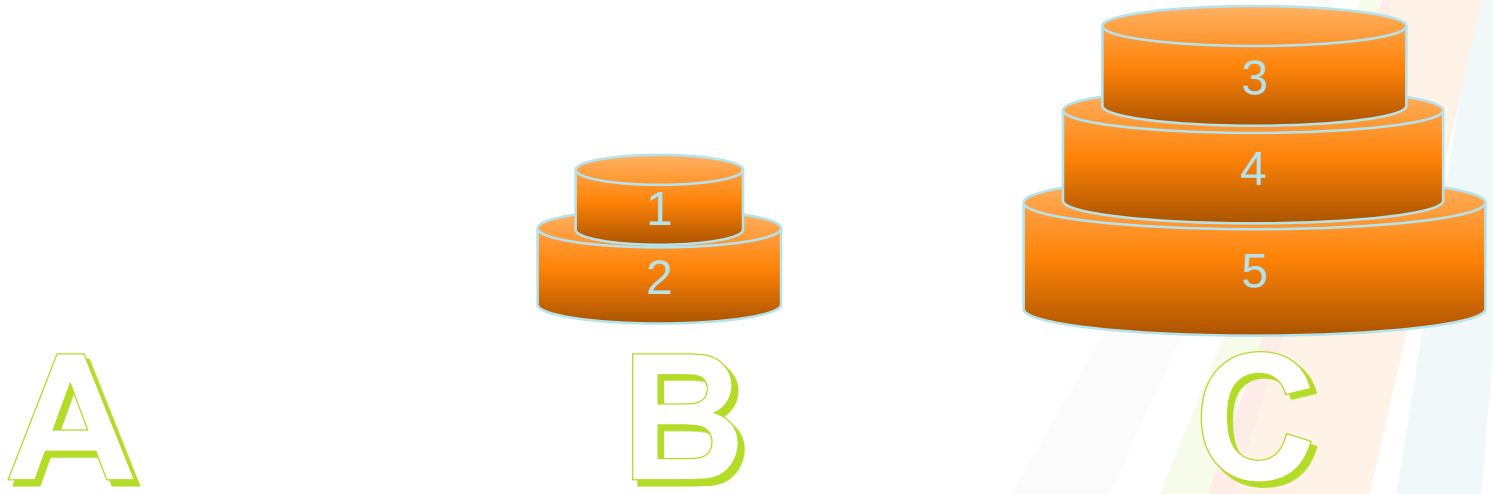
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



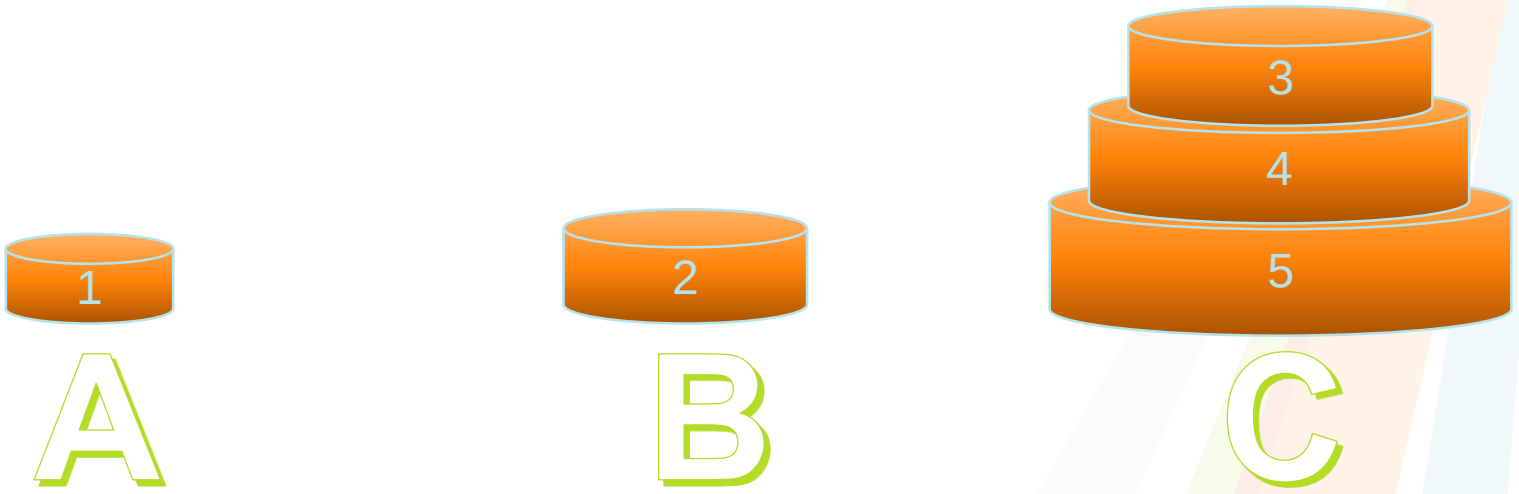
3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội



3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội

3.1. Khái niệm đệ quy

Minh họa bài toán tháp Hà Nội

A B C

3.2. Chiến lược vét cạn (Brute force)

Tư tưởng: Thử tất cả các khả năng xem khả năng nào là nghiệm đúng của bài toán cần giải quyết.

VD 1: tìm phần tử có GTLN/NN trong mảng.

Duyệt qua tất cả các phần tử trong mảng để thử và tìm nghiệm đúng.

VD 2: in ra tất cả các số nguyên tố có 4 chữ số abcd sao cho ab = cd (các số có 2 chữ số).

3.2. Chiến lược vét cạn (Brute force)

```
for(a=1;a<=9;a++)
        for(b=0;b<=9;b++)
                for(c=0;c<=9;c++)
                        for(d=0;d<=9;d++)
if(KTNT(a*1000+b*100+c*10+d) && (10*a+b==10*c+d))
        cout<<a<<b<<c<<d;
```

3.2. Chiến lược vét cạn (Brute force)

Ưu điểm:

- Luôn tìm ra nghiệm chính xác.
- Có thể áp dụng cho mọi loại bài toán.

Nhược điểm:

- Cần phải thử tất cả các khả năng =>chi phí thời gian lớn.
- Phù hợp với dữ liệu đầu vào nhỏ.

3.3. Chiến lược quay lui (Backtracking/try and error)

Tư tưởng:

- Lưu các trạng thái trên đường đi tìm nghiệm của bài toán.
- Tới bước nào đó không thể tiến hành tiếp, sẽ thực hiện thao tác quay lui (back tracking) về trạng thái trước đó.
- Lựa chọn các khả năng khác.

3.3. Chiến lược quay lui (Backtracking/try and error)

Như vậy:

- Dùng để giải bài toán liệt kê các cấu hình
- Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được thử hết tất cả các khả năng có thể.

3.3. Chiến lược quay lui (Backtracking/try and error)

Áp dụng: có thể các trường hợp sau

- Tìm một nghiệm có thể có của bài toán.
- Tìm tất cả các nghiệm sau đó chọn lấy một nghiệm thỏa mãn điều kiện cụ thể nào đó.
- Tìm tất cả các nghiệm của bài toán.

3.3. Chiến lược quay lui (Backtracking/try and error)

- Phương pháp giải quyết:
  - *Giả thiết cấu hình cần liệt kê có dạng (X1,X2,..Xn).* Khi đó thuật toán quay lui được thực hiện qua các bước:
    - Xét tất cả các giá trị X1 có thể nhận, thử cho X1 nhận lần lượt các giá trị đó. Với mỗi giá trị thử cho X1 ta sẽ:
    - Xét tất cả các giá trị X2 có thể nhận, lại thử cho X2 nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho X2 lại xét tiếp các khả năng chọn X3, lại xét tiếp các khả năng chọn X3, cứ tiếp tục như vậy đến bước:
- ……
- n) Xét tất cả các giá trị Xn có thể nhận, thử cho Xn nhận lần lượt các giá trị đó, thông báo cấu hình tìm được (X1, X2,..,Xn)

3.3. Chiến lược quay lui (Backtracking/try and error)

```
Try(k) {
        for([Mỗi phương án chọn i(thuộc tập D)]) {
                if ([Chấp nhận i]) {
                        [Chọn i cho X[k]];
                        if ([Thành công]) {
                                [Đưa ra kết quả];
                        } else {
                                Try(k+1);
                                [Bỏ chọn i cho X[k]];
                        }
                }
        }
}
```

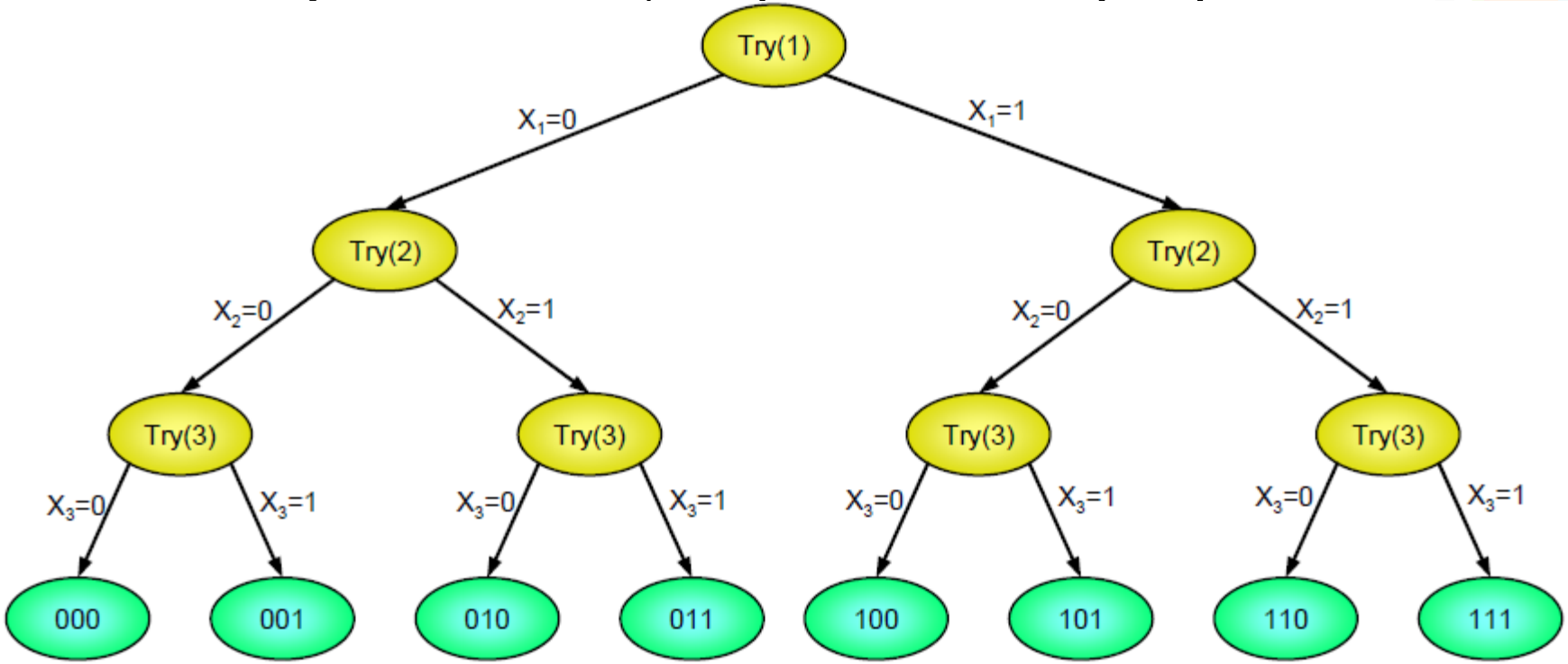
3.3. Chiến lược quay lui (Backtracking/try and error)

- VD1:

Liệt kê các dãy nhị phân có độ dài n

– Ý tưởng quay lui: biểu diễn dãy nhị phân dưới dạng x[1..n]. Mỗi phần tử x[i] có thể nhận 2 giá trị {0,1}. Với mỗi giá trị thử gán cho x[i], tiếp tục thử các giá trị có thể gán cho x[i+1]…

- Liệt kê các dãy nhị phân có độ dài n
  - Ví dụ: Khi n = 3, cây tìm kiếm quay lui như



- Liệt kê các dãy nhị phân có độ dài n
  - Giải thuật

```
Attempt(i)
    for j=0 to 1 do
        x[i] = j
        if i==n-1 then PrintResult
        else Attempt(i+1)
main()
    Attempt(0)
```

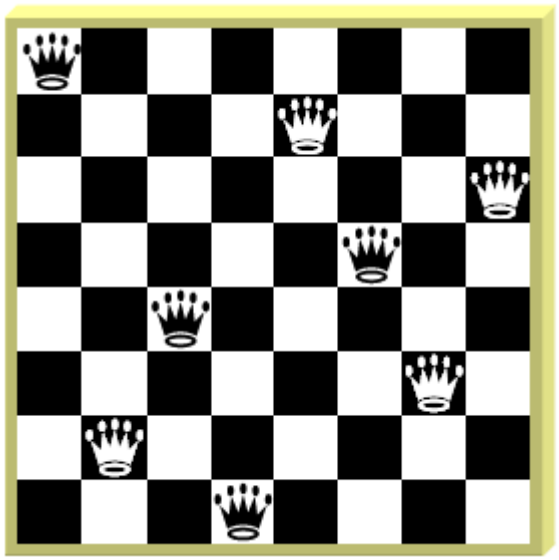
- Liệt kê các chỉnh hợp không lặp chập k
  - Ý tưởng quay lui: biểu diễn các cấu hình dưới dạng x[1..k]. Xét tất cả các khả năng chọn x[i], thử hết các giá trị từ 1 đến n mà giá trị này chưa bị các phần tử đứng trước chọn.
  - Để quản lý việc chọn các phần tử, ta sử dụng kỹ thuật mảng đánh dấu c[1..n]. Nếu c[i] = false tức là i đã bị chọn rồi.

– Giải thuật

```
Attempt(i)
    for j=1 to n do
        if c[j] then
            x[i] = j
            if i==k-1 then PrintResult
            else
                c[j] = false
                Attempt(i+1)
                c[j] = true
main()
    Attempt(0)
```

- Bài toán 8 con hậu
  - Bài toán: Xét bàn cờ kích thước nxn. Một quân hậu trên bàn cờ có thể ăn được các quân khác nằm tại các ô cùng hàng, cùng cột hoặc cùng đường chéo. Tìm cách xếp n quân hậu trên bàn cờ sao cho không quân nào ăn quân nào

- Bài toán 8 con hậu
  - Ví dụ cách xếp với n = 8



3.3. Chiến lược quay lui (Backtracking/try and error)

Ưu điểm: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.

Nhược điểm: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:

- Rơi vào tình trạng "thrashing": quá trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.
- Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.
- Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết được nhánh tìm kiếm sẽ đi vào bế tắc.

3.4. Kỹ thuật băm (hash)

- Băm là quá trình chuyển đổi đầu vào gồm các chữ cái và ký tự có kích thước không cố định để tạo đầu ra có kích thước cố định.
- Quá trình này được thực hiện bằng cách sử dụng các công thức toán học như các hàm băm (được thực hiện dưới dạng các thuật toán băm).

3.4. Kỹ thuật băm (hash)

Các hàm băm khác nhau sẽ tạo ra các kết quả đầu ra có kích thước khác nhau, nhưng kích thước của các kết quả đầu ra có thể nhận được luôn cố định, không đổi.

Ví dụ, thuật toán SHA-256 chỉ có thể tạo ra các kết quả đầu ra có kích thước 256 bit, trong khi thuật toán SHA-1 sẽ luôn tạo ra một kết quả đại diện có kích thước 160-bit.

3.4. Kỹ thuật băm (hash)

Một số ứng dụng của hash:

- Bảng băm: cho phép tra cứu nhanh một bản ghi dữ liệu nếu cho trước khóa của bản ghi đó
- Mật mã học
- Chữ ký số và xác thực
- Kiểm tra sự toàn vẹn của tệp tin
- Xác minh mật khẩu
- Ứng dụng trong Blockchain

Bài tập

1. Liệt kê các xâu nhị phân có độ dài 10 và không có 2 bit 0 đứng liền nhau
2. Tìm số nguyên tố đứng ngay sau số nguyên tố N
3. Phân tích 1 số tự nhiên N ra thừa số nguyên tố (VD: 100 = 2*2*5*5)


Thank You!