HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ET2100

ONE LOVE. ONE FUTURE.

Chương III: Giải thuật và thủ tục đệ quy

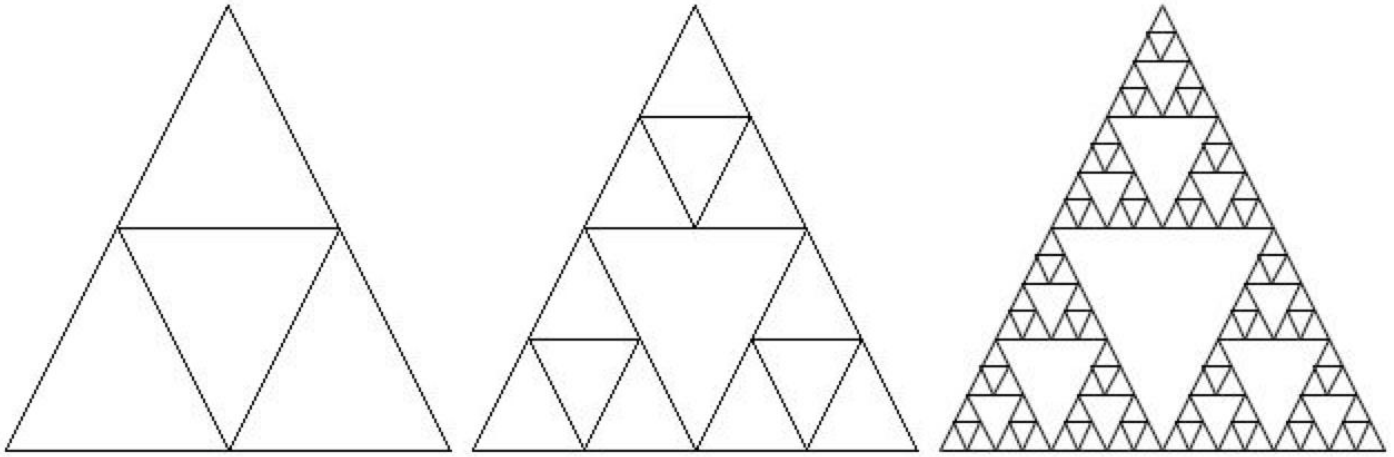
Giảng viên: TS. Đỗ Thị Ngọc Diệp
Khoa Kỹ thuật Truyền thông – Trường Điện Điện Tử

Các nội dung chính

1. Khái niệm về sự đệ quy
   - 1.1. Định nghĩa đệ quy
   - 1.2. Giải thuật đệ quy
2. Xây dựng thủ tục đệ quy
   - 2.1. Thủ tục đệ quy
   - 2.2. Phương pháp xây dựng thủ tục đệ quy
   - 2.3. Sự khử đệ quy
3. Một số ví dụ minh hoạ

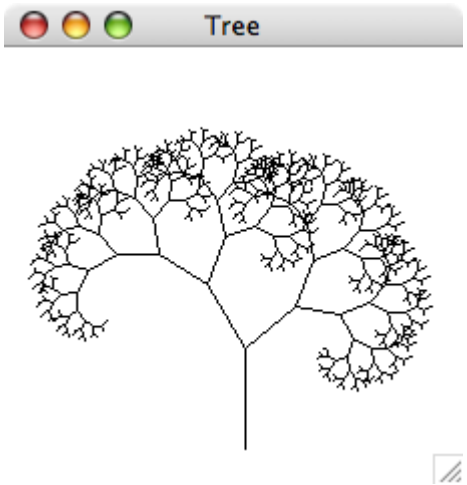
1. Khái niệm về sự đệ quy

1.1. Khái niệm đệ quy



https://runestone.academy

© CanStockPhoto.com - csp4396971



http://www.cburch.com

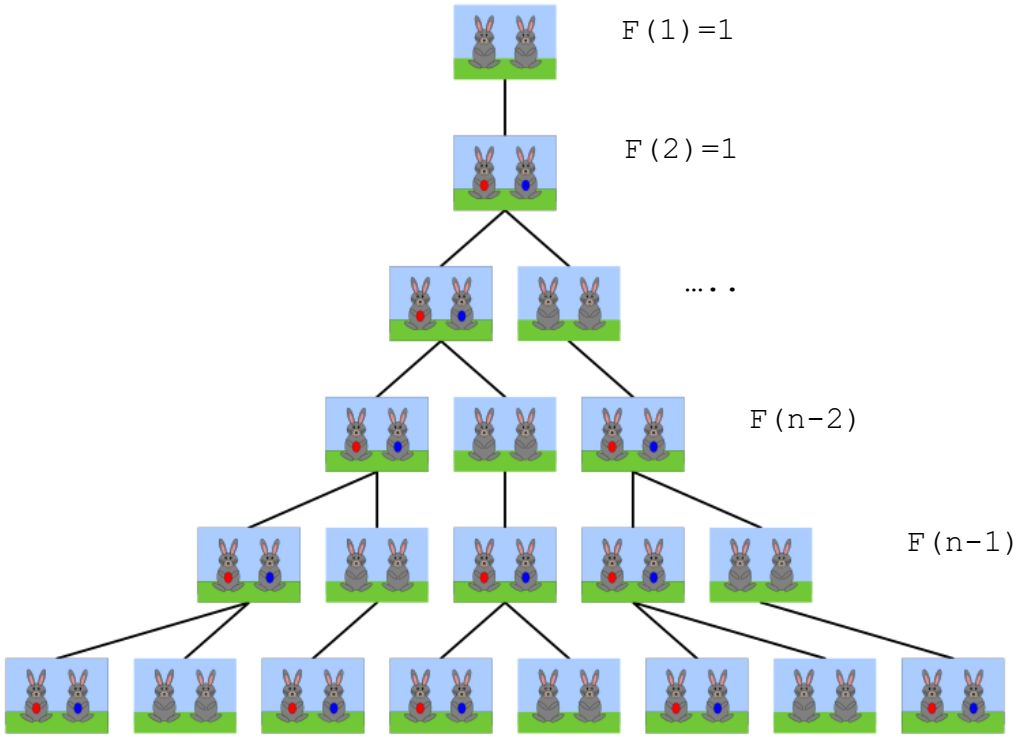
1.1. Khái niệm đệ quy

- Khái niệm: *"ta gọi một đối tượng là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó"*.
- Định nghĩa kiểu đệ quy: là cách định nghĩa một đối tượng/khái niệm dựa vào một hay một tập các đối tượng/khái niệm có **cùng bản chất** với đối tượng/khái niệm cần định nghĩa nhưng có **quy mô nhỏ hơn**.
  - Đệ quy bao gồm:
    - Các quy tắc cơ sở (basic rules)
    - Các quy tắc quy nạp (inductive rules)
- Tính chất đệ quy: các đối tượng/khái niệm có thể định nghĩa một cách đệ quy thì ta gọi chúng có tính chất đệ quy.

- Ví dụ
  - String
    - Quy tắc 1: `String = 1 char + (sub) String`
    - Quy tắc 2: `1 char = String`
  - Số tự nhiên
    - Quy tắc 1: `x ? N if (x-1) ? N`
    - Quy tắc 2: `1 ? N`
  - N!
    - Quy tắc 1: `n! = n * (n-1)!`
    - Quy tắc 2: `0! = 1`
  - Định nghĩa danh sách tuyến tính:
    - Quy tắc 1: Nếu L_{n-1} là một danh sách kích thước `n-1` thì cấu trúc L_n=<a, L_{n-1}> cũng là một DSTT, với `a` là một phần tử có cùng kiểu dữ liệu như các phần tử trong L_{n-1}, và `a` đứng trước L_{n-1} trong danh sách L_n.
    - Quy tắc 2: `L = ∅` là DSTT

- Ví dụ : Dãy số Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, v.v.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, v.v.

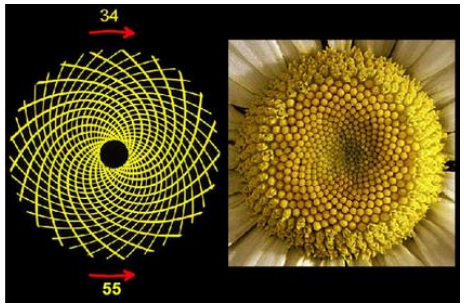


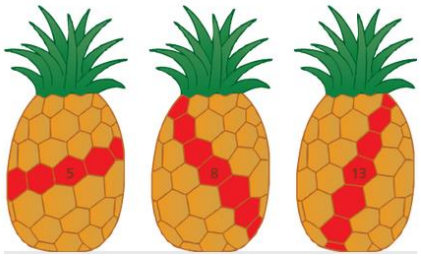
F(n) = ?

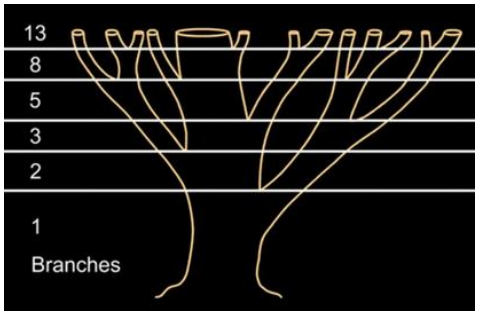
F(n) = F(n-1) + F(n-2)

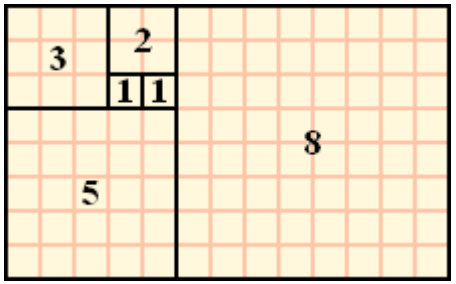
- Ví dụ : Dãy số Fibonacci

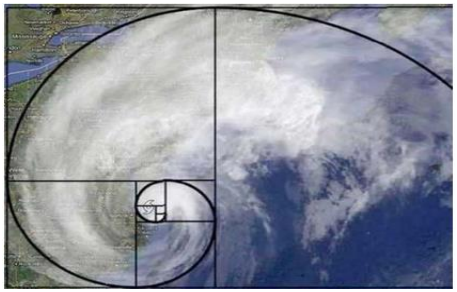
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, v.v.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, v.v.

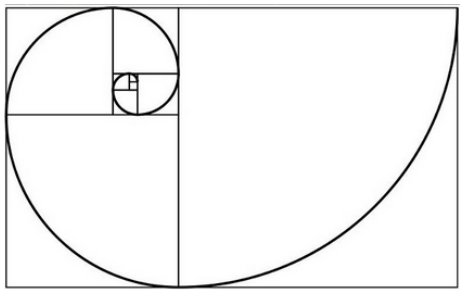


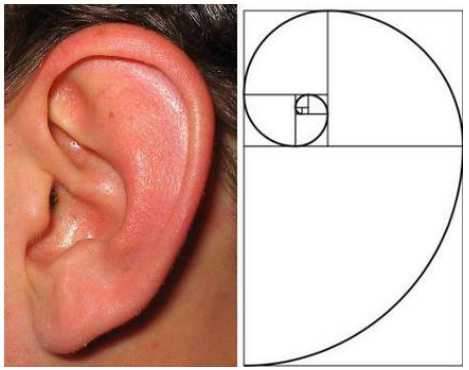




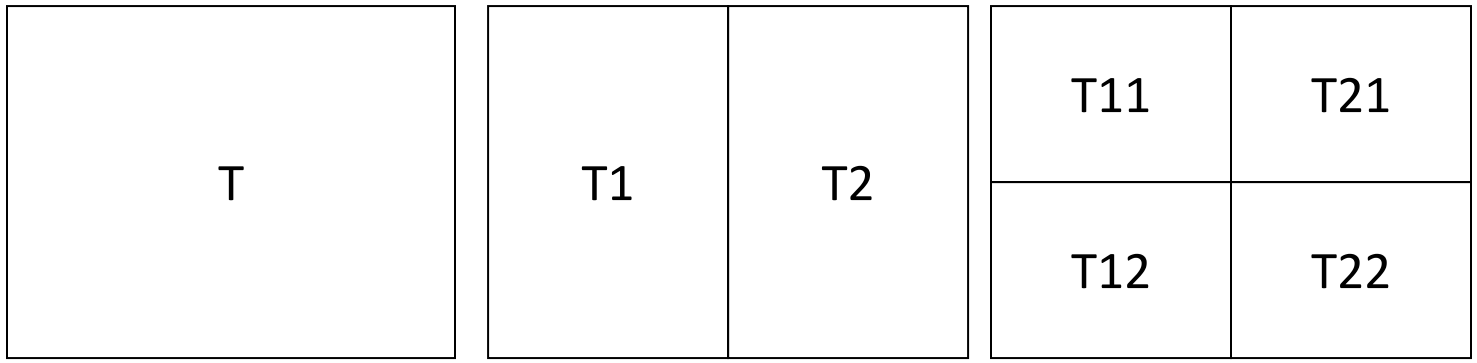




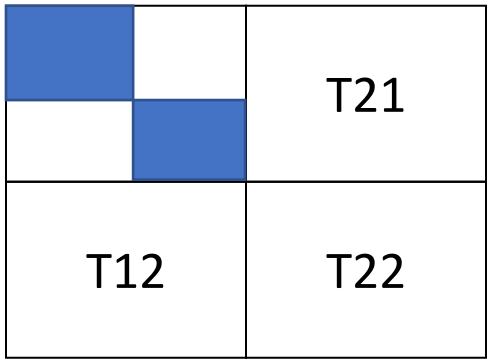




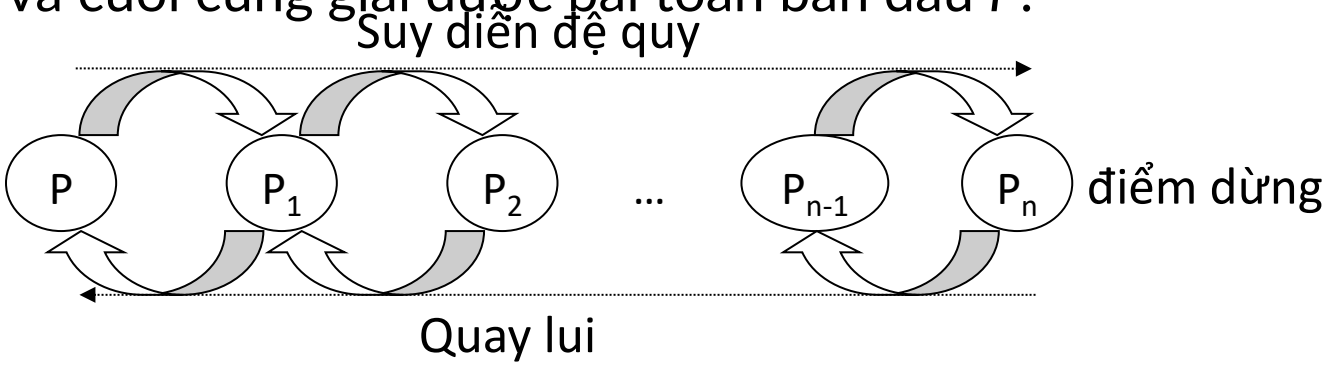
- Ví dụ 1: Tìm *từ* trong *từ điển*



- Sau mỗi lần tách đôi, ta lại dùng đúng giải thuật đã dùng để áp dụng với các phần nhỏ hơn
- Khi chia nhỏ đến 1 trang, việc tìm kiếm sẽ trở nên dễ dàng và có thể thực hiện trực tiếp



- Khái niệm:
  - Đưa bài toán P về một bài toán P_1 có bản chất tương tự P nhưng có quy mô bé hơn.
    - Nghĩa là tồn tại một quan hệ giữa P và P_1, khẳng định nếu giải được P_1 thì sẽ giải được P.
  - Tương tự, đưa P_1 về P_2 có cùng bản chất với P_1, quy mô nhỏ hơn P_1.
  - ...
  - Quá trình cứ tiếp tục cho đến khi ta đưa bài toán về bài toán con P_n tương tự như P_{n-1} và có quy mô nhỏ hơn P_{n-1} .
  - P_n có thể giải một cách trực tiếp do P_n đã đủ nhỏ và đơn giản.
  - Sau khi giải được P_n, ta quay lại giải các bài toán con theo trật tự ngược lại và cuối cùng giải được bài toán ban đầu P.



- Ví dụ 2: Tính giá trị dãy số Fibonacci
  - F(n) = 1 với n = 1, 2;
  - F(n) = F(n-1) + F(n-2) với n > 2;
- Nhận xét:
  - Thay vì tính *F(n)* ta quy về tính các giá trị hàm *F* với biến *n* nhỏ hơn là
    F(n-1) và *F(n-2)*.
  - Cách tính các hàm *F(n-1)*, *F(n-2)* này cũng giống như cách tính *F(n)*, có nghĩa là cũng quy về việc tính các hàm *F* với biến *n* nhỏ hơn nữa
  - Trường hợp đặc biệt: khi *n* đủ nhỏ (*n <= 2*) ta tính được trực tiếp giá trị của hàm *F(1) = 1*, *F(2) = 1*

- Ví dụ 3: Tìm số nhỏ nhất trong một dãy N số $a_1, a_2, ..., a_N$
  - 1) Nếu $N=1$ thì $min=a_1$;
  - 2) Chia dãy ban đầu thành hai dãy con:

    $L_1 = a_1, a_2,..., a_m$ và $L_2 = a_{m+1}, a_{m+2},..., a_N$ với $m=(1+N)\ DIV\ 2$.

    Tìm min_1 trong dãy L_1 và min_2 trong dãy L_2.

    So sánh min_1 và min_2 để tìm ra min của dãy ban đầu

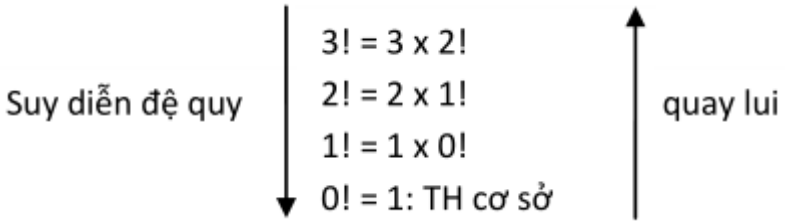
1.2 Giải thuật đệ quy (recursive algorithm)

- Cấu tạo giải thuật đệ quy:
  - *Trường hợp cơ sở*: trường hợp bài toán P_n có quy mô đủ nhỏ để ta có thể giải trực tiếp.
    - Đóng vai trò *điểm dừng* trong quá trình suy diễn đệ quy, và cũng quyết định tính dừng của giải thuật đệ quy.
  - *Trường hợp đệ quy*: trường hợp khái quát chứa cơ chế đệ quy.
    - Cơ chế này được áp dụng nhiều lần để thu nhỏ quy mô bài toán cần giải, từ bài toán ban đầu cho đến bài toán nhỏ nhất ở trường hợp cơ sở.
  - Hai trường hợp trên có mối quan hệ khăng khít để tạo thành giải thuật đệ quy và đảm bảo giải thuật đệ quy có thể giải được và có điểm dừng.

Lưu ý: Trong trường hợp không có trường hợp cơ sở, giải thuật rơi vào vòng lặp vô hạn (không có tính dừng) và bài toán không thể giải được.

1.2 Giải thuật đệ quy (recursive algorithm)

- Hoạt động của giải thuật:
  - *Quá trình suy diễn đệ quy*: là quá trình thu nhỏ bài toán ban đầu về các bài toán trung gian tương tự nhưng có quy mô giảm dần bằng cách áp dụng cơ chế đệ quy, cho đến khi gặp trường hợp cơ sở (điểm dừng của quá trình suy diễn đệ quy).
  - *Quá trình quay lui (hay suy diễn ngược)*: là quá trình từ kết quả thu được trong trường hợp cơ sở, thực hiện giải các bài toán trung gian ở quá trình suy diễn đệ quy *theo trật tự ngược lại*, cho đến khi giải quyết được bài toán ban đầu.
- Ví dụ: tính 3! theo giải thuật đệ quy



2. Xây dựng thủ tục đệ quy

2.1 Thủ tục đệ quy

- *Khái niệm*: cơ chế, công cụ để hỗ trợ công việc cài đặt các giải thuật đệ quy một cách đơn giản và hiệu quả trong các ngôn ngữ lập trình.
- Thủ tục đệ quy là *một chương trình con* trực tiếp cài đặt cho giải thuật đệ quy.
  - Pascal: thủ tục đệ quy, hàm đệ quy
  - C/C++: chỉ có hàm đệ quy
  - …

2.2 Phương pháp xây dựng thủ tục đệ quy

- Bước 1: Tổng quát hóa bài toán
- Bước 2: Xác định các tham số ảnh hưởng tới độ phức tạp của bài toán
- Bước 3: Xác định các trường hợp cơ sở
  - Trường hợp suy biến của tham số
  - Trường hợp tương ứng với các gía trị biên của tham số
- Bước 4: Xây dựng thủ tục có tính chất đệ quy theo tham số đã xác định

- *Cấu trúc* của thủ tục đệ quy
  - Trong thủ tục đệ quy có *lời gọi* đến chính thủ tục đó.
  - Mỗi lần thực hiện gọi lại thủ tục thì kích thước của bài toán lại *thu nhỏ* hơn trước.
  - Có trường hợp *cơ sở* để dừng đệ quy, kết thúc bài toán.

```
void  P(A) {
    if (A==A0) then // giải quyết trường hợp cơ sở
        …
    else {          // trường hợp đệ quy
        …
        P(A1);      // lời gọi đệ quy, A1<A
    …
    }
}
```

- Tính giá trị n!
  - 0! = 1
  - n! = n * (n-1)!

```
int Fact (int n){
    if  (n <= 1) return 1;
    else return n * Fact (n-1);
}
```

- Tính giá trị dãy Fibonacci
  - F (n) = 1 với n <= 2;
  - F (n) = F(n-1) + F(n-2) với n > 2;

```
int Fibo1 (int n) {
    if (n <= 2) return 1;
    else return (Fibo1 (n-1) + Fibo1 (n-2));
}
```

2.2 Phương pháp xây dựng thủ tục đệ quy

- Các loại đệ quy:
  - Đệ quy trực tiếp
    vs. Đệ quy gián tiếp
    (đệ quy tương hỗ)

```
int Func (int n)
{
    if (n == 0)
        return n;
    else
        return (Func (n-1));
}
```

```
int Func1 (int n)
{
    if (n == 0)
        return n;
    else
        return Func2(n);
}
int Func2(int x)
{
     return Func1(x-1);
}
```

  - Đệ quy đuôi (*tail recursion*)
    vs. Đệ quy không đuôi (*non-tail recursion*)

```
return Fact1(n-1, n*res);
```

```
return (n * Fact(n-1));
```

  - Đệ quy tuyến tính *Fact*
  - Đệ quy nhị phân *Fibonacci*
  - Đệ quy dạng cây
  - Đệ quy phi tuyến
  - Đệ quy lồng (nested recursion)

```
int ackerman(int m, int n)
{
    if (m == 0)
        return (n + 1);
    else
        if (n == 0)
            return ackerman(m - 1, 1);
        else
            return ackerman(m - 1, ackerman(m, n - 1));
}
```

- Hoạt động của thủ tục đệ quy
  - Quá trình suy diễn đệ quy ⇔ Giai đoạn gọi đệ quy:
    - Bắt đầu từ lời gọi thủ tục đầu tiên
    - CT sẽ đi theo nhánh gọi đệ quy trong thân thủ tục để liên tục gọi các lời gọi đệ quy trung gian cho đến khi gặp trường hợp cơ sở
    - Khi đến điểm dừng này, hệ thống sẽ tự động kích hoạt giai đoạn thứ hai là giai đoạn quay lui
  - Quá trình quay lui ⇔ Giai đoạn quay lui:
    - Hệ thống sẽ thi hành các thủ tục đệ quy trung gian trong giai đoạn đầu theo thứ tự ngược lại
    - cho đến khi thi hành xong thủ tục được gọi đầu tiên thì kết thúc
    - đồng thời kết thúc hoạt động của thủ tục đệ quy

- Vấn đề cài đặt: lưu trữ, xử lý các lời gọi đệ quy
  - sử dụng cấu trúc ngăn xếp để lưu trữ các lời gọi đệ quy.
  - Thực tế, các NNLT có hỗ trợ cài đặt thủ tục đệ quy đều đã tự động cài đặt các cấu trúc ngăn xếp thích hợp

Giải thuật đệ quy

Suy diễn đệ quy

3! = 3 x 2!
2! = 2 x 1!
1! = 1 x 0!
0! = 1: TH cơ sở

quay lui

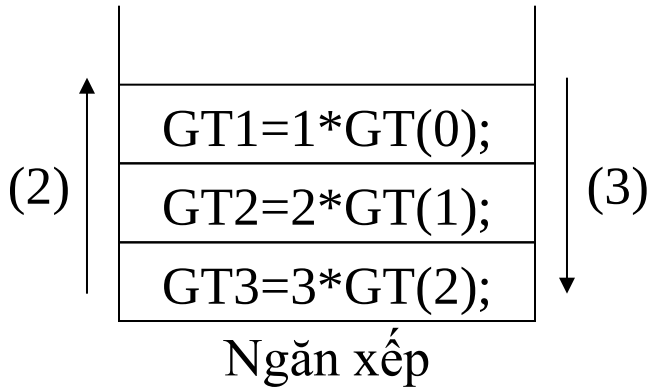
Hoạt động của thủ tục đệ quy

(1) Khởi tạo ngăn xếp
(2) Giai đoạn gọi đệ quy
(3) Giai đoạn quay lui

(1) Gọi 3!: GT(3);

"Độ sâu đệ quy"

RecursionError: maximum recursion depth exceeded



2.3. Sự khử đệ quy

- Khái niệm:
  - Khi thay các giải thuật đệ quy bằng các giải thuật không tự gọi chúng, ta gọi đó là ***sự khử đệ quy***
  - Sự khử đệ quy thông qua các vòng lặp (*for*, *while*) và phân nhánh (*if…then…else*), lưu trữ trạng thái trung gian
  - Khử đệ quy không phải bao giờ cũng dễ thực hiện

Khử đệ quy bằng vòng lặp

Đệ quy đuôi

```
if (TH cơ sở) {
    ...
} else {
    Gọi đệ quy (phạm vi mới)
}
```

+ Nối tiếp các lời gọi

```
int USCLN(int m, int n) {
    if (n == 0) return m;
    else return USCLN(n, m % n);
}
```

```
int USCLN(int m , int n ){
    while(n != 0 ) {
        int temp = m % n ;
        m = n ;
        n = temp ;
    }
    return m ;
}
```

Khử đệ quy bằng vòng lặp

Dạng đệ quy không đuôi:
+ Trường hợp thường gặp

```
if (n<=n0) {
    Trả về giá trị X
} else {
    Gọi đệ quy f(n*: phạm vi mới)
    Tính toán giá trị trả về = g(n,f(n*))
}
```

+ Xác định được bước nhảy trong mỗi lần lặp, xác định được số lần lặp của đệ quy
+ Triển khai theo hướng ngược lại

```
int Fact (int n){
    if  (n <= 1) return 1;
    else return n * Fact (n-1);
}
```

```
int Fact (int n){
    if (n <= 1) return 1;
    else {
        int x=1;
        for (int i=2;i<=n;i++) x*=i;
        return x;
    }
}
```

Khử đệ quy bằng vòng lặp

```
int Fibo (int n){
    if (n <= 2) return 1;
    else
       return (Fibo1 (n-1) + Fibo1 (n-2));
}
```

+ Xác định được bước nhảy trong mỗi lần lặp, xác định được số lần lặp của đệ quy
+ Triển khai theo hướng ngược lại

```
int  Fibo (int n) {
    int i, f, f1, f2;
    f1 = 1 ;
    f2 = 1 ;
    if  (n <= 2)  f = 1;
    else
      for (i = 3;i<=n;i++){
         f = f1 + f2;
         f1 = f2;
         f2 = f;
      }
    return f;
}
```

Khử đệ quy bằng Stack

- Giai đoạn đệ quy: Lưu thông tin trung gian ở từng bước vào Stack
- Gặp trường hợp cơ sở → chuyển sang GĐ quay lui
- Giai đoạn quay lui: Khôi phục lại thông tin ở thời điểm trước đó bằng cách lấy từ đỉnh Stack

```
int Fact (int n){
    if  (n <= 1) return 1;
    else return n * Fact (n-1);
}
```

```
int Fact2(int n) {
    std::stack<int> S;
    while (n>1) {
        S.push(n);
        n=n-1;
    }
    int Fact = 1;
    while(!S.empty()){
        n=S.top();
        S.pop();
        Fact *= n ;
    }
    return Fact;
}
```

(1) Khác TH cơ sở

(2) Lưu thông tin *n*

(3)Thu hẹp phạm vi

(4)Lặp lại quá trình

(5)Xử lý TH cơ sở

(6) Khôi phục thông tin

Khử đệ quy bằng Stack

```
int USCLN(int m, int n) {
    if (n == 0) return m;
    else return USCLN(n, m % n);

}
```

```
// if n==0: USCLN = m;
// else { temp = m%n; m=n; n=temp;
// call again USCLN(m,n); }
```

```
int USCLN(int m, int n) {
    std::stack<int> S;
    while (n != 0) {
        S.push(m); S.push(n);
        int temp = m%n; m=n; n=temp;
    }
    int Result = m;
    while(!S.empty()){      // can be removed
        S.pop(); S.pop();   // for nothing
    }
    return Result;
}
```

(1) Khác TH cơ sở

(2) Lưu thông tin *m, n*

(3)Thu hẹp phạm vi

(4)Lặp lại quá trình

(5)Xử lý TH cơ sở

(6) Khôi phục thông tin

So sánh Đệ quy vs. Lặp !

```
int Fact1 (int n){
        if  (n <= 1) return 1;
        else return n * Fact1 (n-1);
}

int Fact2(int n){
   int prod = 1;
   while (n>1) prod *= n--;
   return prod;
}
```

```
int Fact3(int n) {
   std::stack<int> S;
   while (n>1) {
        S.push(n); n=n-1; }
   int Fact = 1;
   while(!S.empty()){
        n=S.top(); S.pop();
        Fact *= n ;
   }
   return Fact;
}
```

Rescursive: 7! = 5040; time = 0.000744 s
Iterative: 7! = 5040; time = 0.000370 s
Stack: 7! = 5040; time = 0.020343 s

```
int main(){
   clock_t  t0, t1;
   int n = 7, k = 30000, i;
   int f;
   t0 = clock();    for(i=0;i<k;i++) f = Fact1(n);    t1 = clock();
      printf("Rescursive: %d! = %d; time = %f s \n", n, f, (float)(t1-t0) / CLOCKS_PER_SEC);
   t0 = clock();    for(i=0;i<k;i++) f = Fact2(n);    t1 = clock();
      printf("Iterative: %d! = %d; time = %f s \n", n, f, (float)(t1-t0) / CLOCKS_PER_SEC);
   t0 = clock();    for(i=0;i<k;i++) f = Fact3(n);    t1 = clock();
      printf("Stack: %d! = %d; time = %f s \n", n, f, (float)(t1-t0) / CLOCKS_PER_SEC);
   return 0;
}
```

So sánh Đệ quy vs. Lặp !

```
int Fibo1 (int n){
   if (n <= 2) return 1;
   else return (Fibo1 (n-1) + Fibo1 (n-2));
}
int  Fibo2 (int n) {
   int i, f, f1, f2;
   f1 = 1 ;  f2 = 1 ;
   if  (n <= 2)  f = 1;
   else for (i = 3;i<=n;i++){
       f = f1 + f2;
       f1 = f2;
       f2 = f;
   }
   return f;
}
```

// k = 30000

Recursive: Fibo(17) = 1597; time = 0.265261 s

Iteration: Fibo(17) = 1597; time = 0.001370 s

Nhận xét

- Ưu nhược điểm thủ tục đệ quy
  - Ưu điểm: viết chương trình dễ dàng, dễ hiểu, ngắn gọn
  - Nhược điểm:
    - Tốn thời gian, ốn bộ nhớ và có thể tràn stack nếu không kiểm soát tốt độ sâu của đệ quy
    - Khó gỡ lỗi
  - Không phải lúc nào cũng có thể xây dựng bài toán theo giải thuật và thủ tục đệ quy một cách dễ dàng
    - Có thể định nghĩa bài toán dưới dạng bài toán cùng loại nhưng nhỏ hơn như thế nào?
    - Làm thế nào để đảm bảo kích thước bài toán giảm đi sau mỗi lần gọi?
    - Xem xét và định nghĩa các trường hợp cơ sở (trường hợp suy biến) như thế nào?
  - Ứng dụng phổ biến:
    - Thuật toán Tìm kiếm và Sắp xếp
    - Thuật toán trên cây (tree) và đồ thị (graph)
    - Quy hoạch động
    - v.v.

3. Các ví dụ minh họa

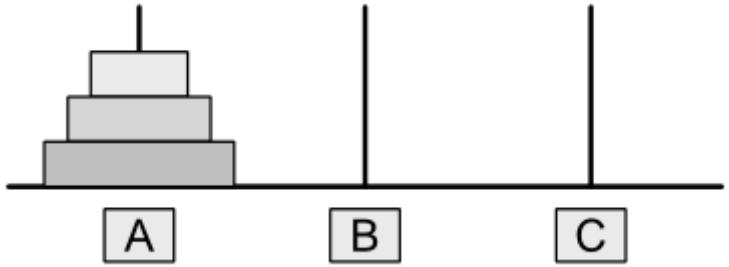
1. Tìm kiếm trong danh sách

- Tìm kiếm một phần tử *x* trong danh sách *H* (con trỏ đến nút đầu tiên). Trả về con trỏ trỏ vào nút tìm thấy. Nếu không tìm thấy thì trả về NULL.

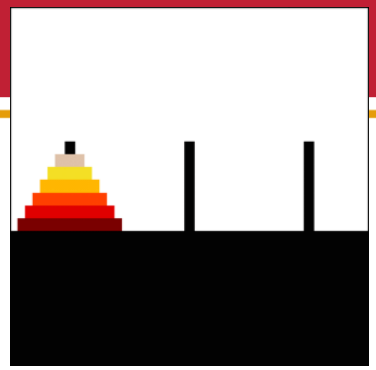
```
PNode Search (Item x, PNode H) {
    if (H == NULL) return NULL;
    else
     if  (H->info == x) return H;
     else  return Search (x, H->next) ;
}
```

2. Bài toán Tháp Hà Nội

- Bài toán: có 3 cột A, B, C. Chuyển cột đĩa từ A → C
  - Mỗi lần chỉ được chuyển 1 đĩa trên cùng
  - Không được đĩa to trên đĩa nhỏ, dù chỉ tạm thời



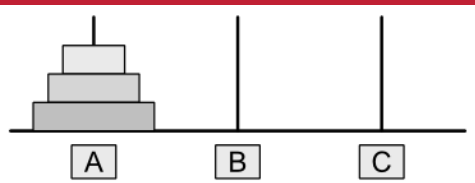
Bài toán Tháp Hà Nội



- Cách giải 1:
  - Giải pháp lặp
    - Số đĩa chẵn:
      - thực hiện 01 di chuyển hợp pháp giữa A và B (theo cả hai hướng)
      - thực hiện 01 di chuyển hợp pháp giữa A và C (theo cả hai hướng)
      - thực hiện 01 di chuyển hợp pháp giữa B và C (theo cả hai hướng)
      - Lặp lại cho đến khi hoàn thành.
    - Số đĩa lẻ:
      - thực hiện 01 di chuyển hợp pháp giữa A và C (theo cả hai hướng)
      - thực hiện 01 di chuyển hợp pháp giữa A và B (theo cả hai hướng)
      - thực hiện 01 di chuyển hợp pháp giữa B và C (theo cả hai hướng)
      - lặp lại cho đến khi hoàn thành.
    - Trong mỗi trường hợp, có tổng cộng 2^n - 1 lần di chuyển được thực hiện.

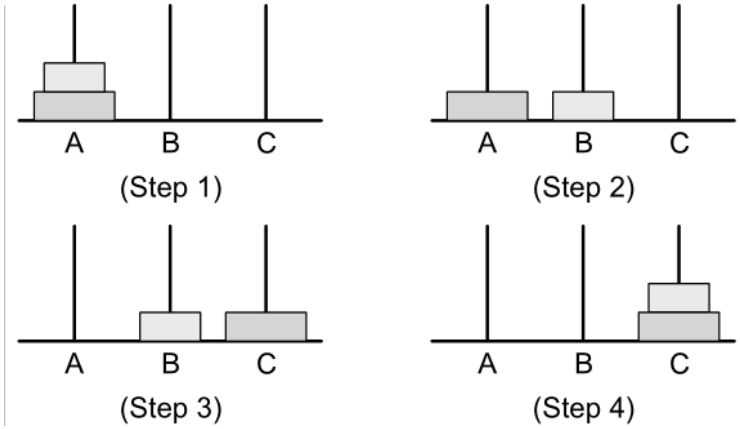
https://en.wikipedia.org/wiki/Tower_of_Hanoi

Bài toán Tháp Hà Nội

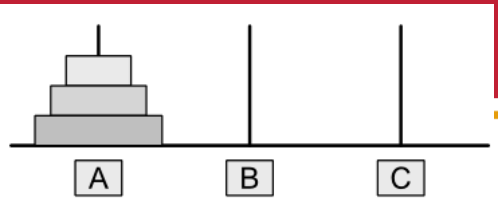


- Cách giải 2: Đệ quy
  - Xác định trường hợp đơn giản (suy biến)
    - n = 1 đĩa: chuyển A sang C [dùng B làm trung gian]
  - Trường hợp đệ quy:
    - B1. Chuyển (n-1) đĩa từ A sang B, dùng C làm trung gian
    - B2. Chuyển 1 đĩa còn lại ở A sang C, dùng B làm trung gian
    - B3. Chuyển (n-1) đĩa từ B sang C, dùng A làm trung gian

n=2



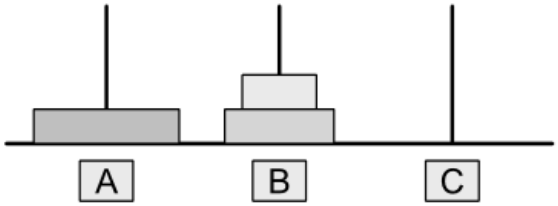
Bài toán Tháp Hà Nội



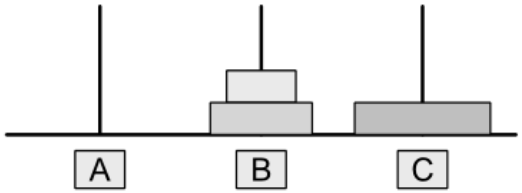
- Cách giải 2: Đệ quy

n=3

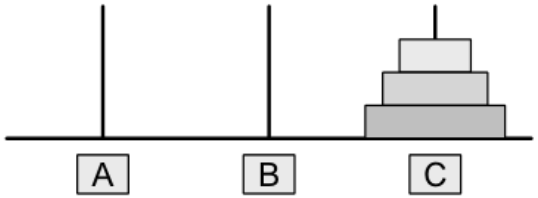
- n=2, A sang B



- 1 đĩa ở A sang C

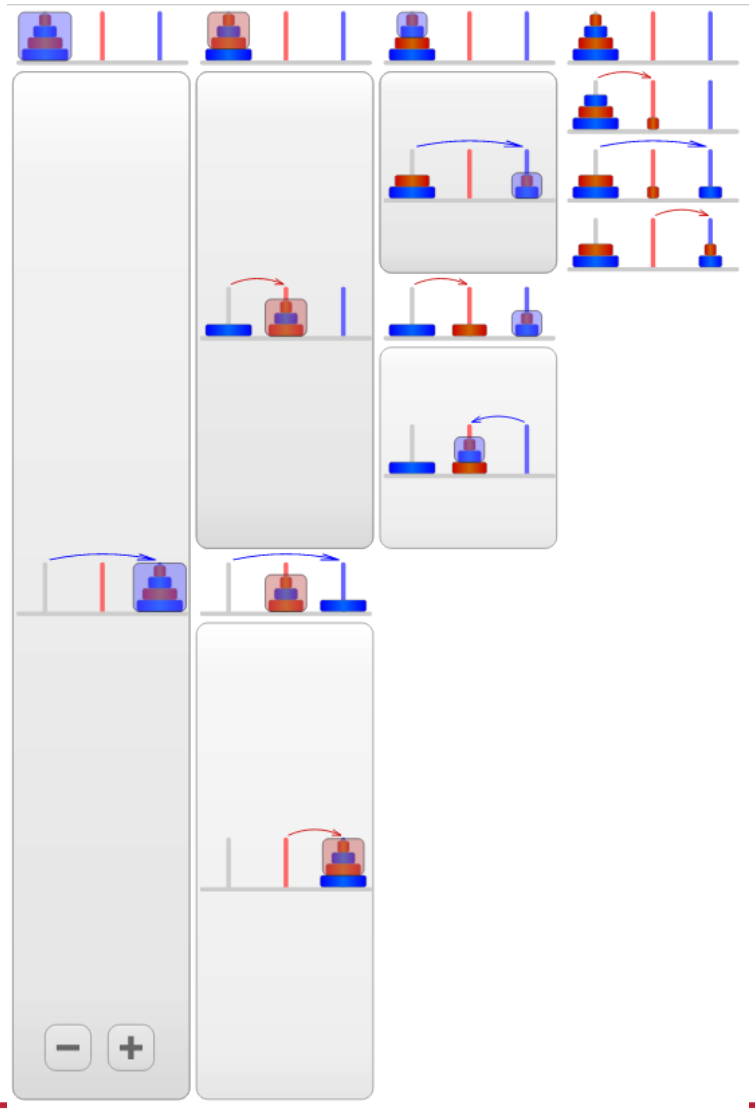


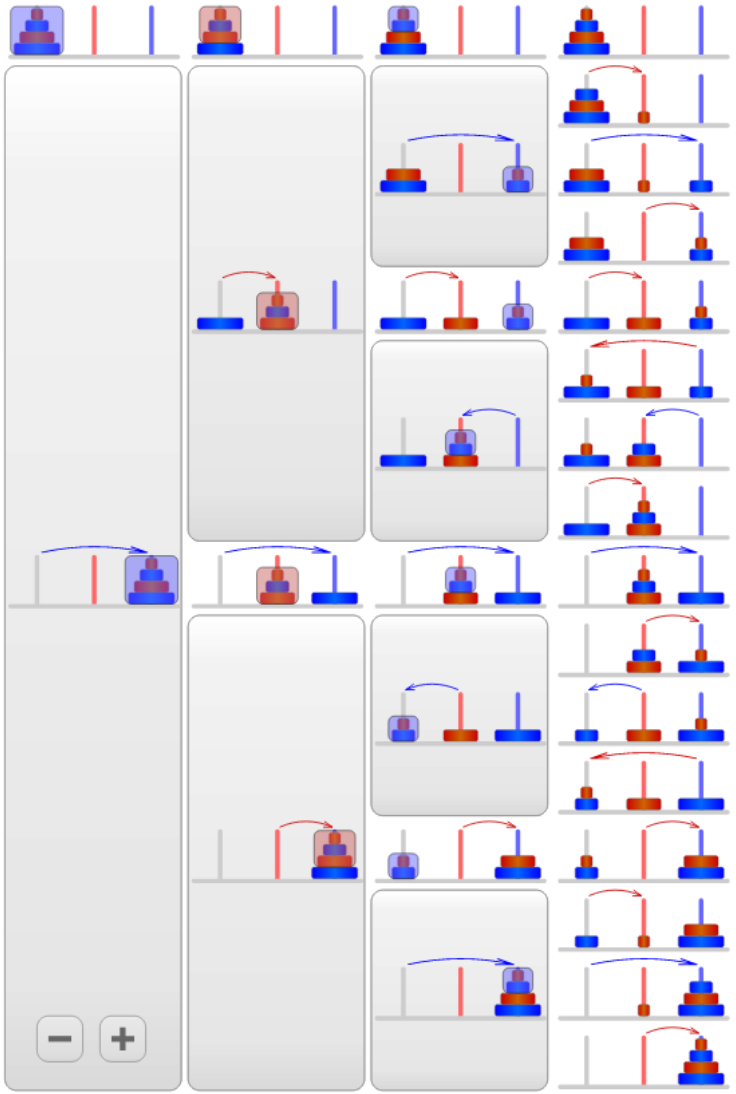
- n=2, B sang C



Bài toán Tháp Hà Nội

n=4





https://en.wikipedia.org/wiki/Tower_of_Hanoi

Bài toán Tháp Hà Nội

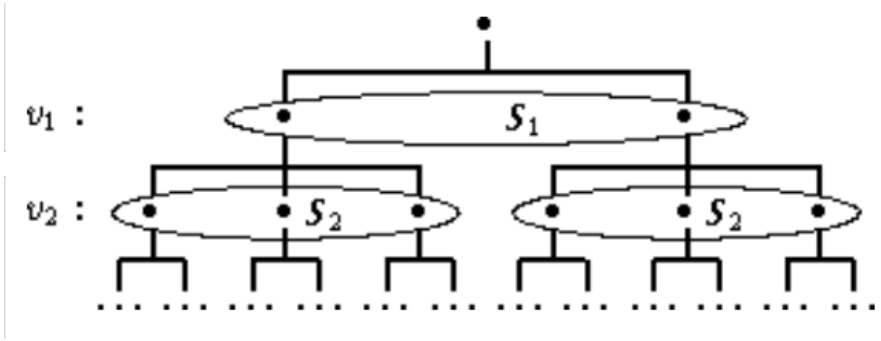
- Giải thuật đệ quy

```
void move(int n, char source, char dest, char spare)
{
  if (n==1)
        printf("\n Move from %c to %c",source,dest);
  else
  {
        move(n-1,source,spare,dest);
        move(1,source,dest,spare);
        move(n-1,spare,dest,source);
  }
}
```

3. Giải thuật quay lui (Backtracking)

- Bài toán: Cho tập N biến $x_1, x_2,..., x_N$ trong miền giá trị tương ứng $S_1, S_2,..., S_N$. Tìm các bộ giá trị của N biến thỏa mãn điều kiện K cho trước.
- Ý tưởng giải thuật quay lui:
  - "*searches for a solution among all available options*"
  - Thực hiện duyệt tất cả các giá trị trong miền $S_1, S_2,..., S_N$ cho đến khi tìm thấy vector $v=<v_1, v_2, ..., v_N>$ thỏa mãn điều kiện K.
  - ...

- Ý tưởng giải thuật:
  - ...
  - Ban đầu v = <rỗng>
  - Ở mỗi bước, mở rộng véc tơ v với một giá trị mới.
  - Khi mở rộng đến <v_1,...,v_i> không thể giúp thỏa mãn điều kiện K, thuật toán sẽ quay lui bằng cách xóa giá trị cuối cùng khỏi vectơ, sau đó tiếp tục mở rộng vectơ bằng các giá trị thay thế khác.
  - Cứ như vậy đến khi tìm được các vec tơ <v_1,v_2,...,v_N> hoặc không



- Giải thuật đệ quy cho quay lui:
  - Giả sử đã thực hiện xong *i* bước đầu tiên nghĩa là ta tìm được *Vi=<x1, x2,..., xi>* thỏa mãn K.
  - Nếu *i=N* kết thúc thuật toán trả về V_N.
  - Nếu không, tiếp tục bước tiếp theo với hai trường hợp :
    - Nếu $\exists p \in S_{i+1}$ phù hợp nghĩa là với $x_{i+1}=p$ thì V_{i+1} thỏa mãn K → Gán $x_{i+1}=p$. Thực hiện bước tiếp theo *i+1* (đệ quy).
    - Ngược lại, nếu $\forall p \in S_{i+1}$ đều có V_{i+1} không thỏa mãn *K* → quay lại bước *i* để thử giá trị phù hợp khác của x_i.
    - Cuối cùng, nếu không tìm thấy và *i=1*, thuật toán kết thúc mà không có vectơ phù hợp nào.

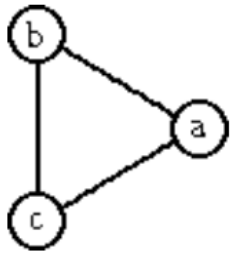
- Giải thuật đệ quy cho quay lui:

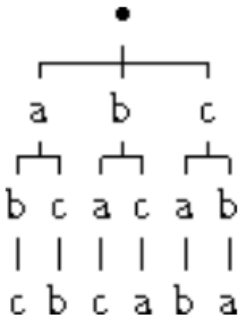
```
void  RBacktrack(vector V, int step)
{
   if (step == N) {
      V is a solution;
   }
   else
      while (exists v_i in S_i and V+{v_i} satisfies K)
      {
         S_i = S_i \ {v_i} ;
         V = V + {v_i};
         RBacktrack(V, step+1);
         V = V \ {v_i};          //prepare before backtrack
      }
}
```

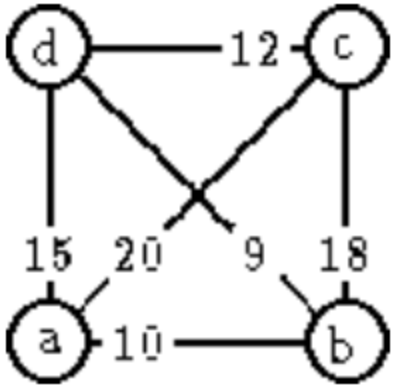
- Giải thuật khử đệ quy cho quay lui:

```
int  NBacktrack (){
    int  i = 1;
    int tong = 0;       //total number of solutions
    vector  V = ∅ ;     //V is empty
    do {
        while (exists v_i in S_i and V+{v_i} satisfies K) {
            S_i = S_i \ {v_i} ;
            V = V + {v_i};
            if (i == N)  {
                V is a solution;
                tong++;
            } else  i++;
        }
        V = V \ {V_i};
        i-- ;                  //Backtrack
    } while  (i >= 1);
    return  tong;
}
```

- Bài toán người bán hàng
  - Giả sử có một tập hợp *n* thành phố và một nhân viên bán hàng cần phải đến mỗi thành phố chính xác một lần và quay trở lại thành phố ban đầu sau khi kết thúc.
  - Tìm các tuyến đường
  - Tìm tuyến đường có độ dài tối thiểu.





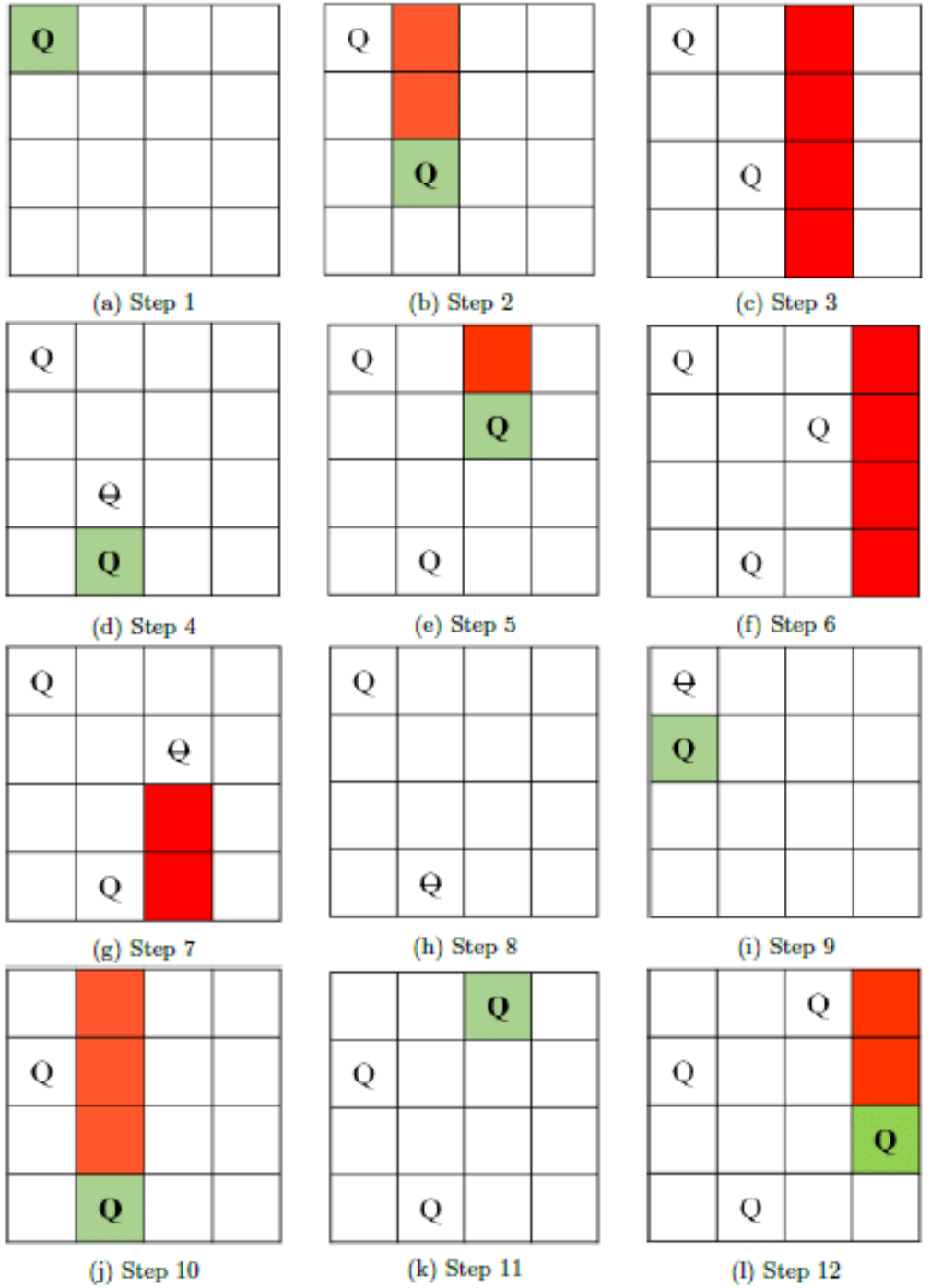


(a, b, d, c) = 51

3. Giải thuật quay lui

- Bài toán sắp quân hậu
  - Xét một bàn cờ *n x n*, tìm cách đặt *n* quân hậu trên bàn cờ mà không có quân hậu nào ăn nhau.

n = 4



Ayub, Ahsan & Kalpoma, Kazi & Proma, Humaira & Kabir, Syed & Chowdhury, Rakib. (2017). Exhaustive study of essential constraint satisfaction problem techniques based on N-Queens problem. 1-6. 10.1109/ICCITECHN.2017.8281850.

Bài tập

- Bài 1: Viết thủ tục đệ quy tính tổng của các phần tử trong một mảng
- Bài 2: Viết thủ tục đệ quy tính tổng các chữ số biểu diễn một số nguyên có n chữ số
- Bài 3: Viết thủ tục đệ quy tìm số nhỏ nhất trong một dãy N số.
- Bài 4: Viết thủ tục đệ quy tìm ước chung lớn nhất của 2 số nguyên
- Bài 5: Viết thủ tục đệ quy tính x^y

Khử đệ quy

```
#include <stdio.h>
int s=0;
void sum(void){
    int x;
    if(scanf("%d",&x)==1){
        s+=x ; sum();
    }
}
int main(){
    puts("Enter a sequence of integers.
            Stop by a non-numeric character\n");
    sum();
    printf("Sum of sequence: %d", s);
    return 0;
}
```

```
void sum(void){
    int x;
    while (scanf("%d",&x)==1){
        s+=x ;
     }
}
```

So sánh Đệ quy vs. Lặp !

- Hàm đệ quy đích thực
  - Hàm tự gọi chính mình nhiều hơn 1 lần.
  - Có thể khó thay thế bởi vòng lặp

```
#include <stdio.h>
void p(int n){
  if(n>0) {
    p(n-2);
    printf("%3d",n);
    p(n-1);
  }
}
```

```
int main(){
  p(4);
  return 0;
}
```