Chương 2. Thuật toán cài đặt đệ quy

Đệ quy, cây đệ quy, định lý thợ, back tracking

Nội dung

- Khái niệm thuật toán cài đặt đệ quy
- Đánh giá thuật toán đệ quy
- Một số chú ý với thuật toán đệ quy
- Thuật toán quy lui – back tracking
- Bài tập

Thuật toán đệ quy

- Thuật toán có thể được cài đặt/biểu diễn theo 2 cách
  - Dùng vòng lặp: dùng các vòng lặp for, do..while, while
  - Dùng đệ quy: Dùng lời gọi đệ quy trong phần cài đặt
- Thuật toán được cài đặt đệ quy (gọi tắt là thuật toán đệ quy)
  - Thường tồi hơn dùng vòng lặp: Máy tính được thiết kế tự nhiên xử lý các lệnh dạng lặp chứ không phải dạng đệ quy
  - Thời gian gọi đệ quy cũng phải được tính vào thời gian thực hiện
  - Đệ quy thường ngắn gọn hơn vòng lặp
  - Đệ quy khó gỡ lỗi hơn vòng lặp
  - Nếu số lần gọi đệ quy quá lớn có thể dẫn đến STACK OVER FLOW
  - Không phải mọi thuật toán đều có thể cài đặt đệ quy

Thuật toán đệ quy

- Định nghĩa đệ quy: *Đối tượng bao gồm chính nó hoặc được định nghĩa dưới dạng chính nó*.

- $n! = \begin{cases} 1 & nếu\ n = 0 \\ n \times (n-1)! & nếu\ n > 0 \end{cases}$
- $Fib(n) = \begin{cases} 1 & nếu\ n = 1, 2 \\ Fib(n-1) + Fib(n-2) & nếu\ n > 2 \end{cases}$
- $P(n) = \begin{cases} 0 & nếu\ n = 0 \\ 1 & nếu\ n = 1 \\ 2P(n-1) + P(n-2) & nếu\ n > 1 \end{cases}$

Elena Filippova. Recursion. 2003
Acrylic on Canvas, 160 cm x 160 cm (63" x 63")

Thuật toán đệ quy

- Mọi định nghĩa đệ quy đều gồm 2 phần
  - Một trường hợp cơ sở (nhỏ nhất) có thể xử lý trực tiếp mà không cần đệ quy, và
  - Một phương thức tổng quát mà biến đổi một trường hợp cụ thể về các trường hợp nhỏ hơn. Do đó biến đổi các trường hợp cho đến khi về trường hợp cơ sở.

Thuật toán đệ quy: Thuật toán có chứa lời gọi đệ quy đến chính nó với đầu vào kích thước nhỏ hơn.

Thuật toán đệ quy

- VD1 . Tính tổng các phần tử trong dãy A gồm n phần tử

```
int sum_loop(int *A, int n)
{
      int sum = 0;
      for (int i = 0; i < n; i++)
            sum = sum + A[i];
      return sum;
}
```

```
int sum_rec(int *A, int n)
{
      if (n <= 0) return 0;
      return A[n - 1] + sum_rec(A, n - 1);
}
```

Thuật toán đệ quy:
- Trường hợp cơ sở: Tính được trực tiếp mà không cần gọi đệ quy
- Thường hợp tổng quát: Chứa lời gọi đệ quy với bài toán con nhỏ dần về trường hợp cơ sở

Thuật toán đệ quy

- VD2. Thuật toán tìm kiếm nhị phân

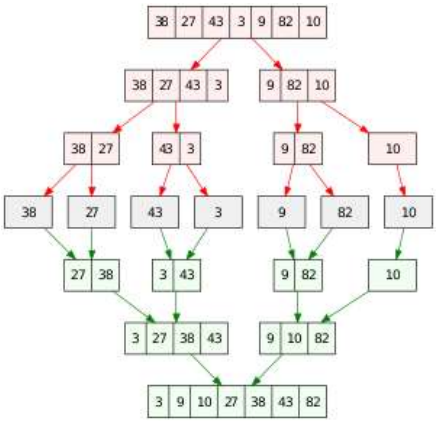
  Cho dãy n phần tử là khóa đã được sắp theo thứ tự tăng dần, và 1 giá trị khóa k. Hãy tìm xem k có xuất hiện trong dãy ban đầu hay không

```
int binSearch_loop(int *A, int n, int key)
{
   int start = 0, end = n - 1, mid;
   while (start <= end)
   {
      mid = (start + end) / 2;
      if (A[mid] == key) return mid;
      else if (A[mid] > key) end = mid - 1;
      else start = mid + 1;
   }
   return -1;
}
```

```
int binSearch_rec(int *A, int start, int end, int key)
{
      if (start > end) return -1;
      int mid = (start + end) / 2;
      if (A[mid] == key) return mid;
      if (A[mid] > key) return binSearch_rec(A,start,mid - 1,key);
      return binSearch_rec(A, mid + 1, end, key);
}
```


Thuật toán đệ quy

- VD 3. Thuật toán sắp xếp trộn – MergeSort




```
MergeSort(int A[], int start, int end)
{
   if(start<end)
   {
          int mid = (start+end)/2;
          MergeSort(A, start, mid);
          MergeSort(A, mid+1, end);
          Merge(A,start,mid,end);
   }
}
```

Hàm Merge trộn 2 danh sách với thời gian O(n)

Thuật toán đệ quy

- Đánh giá thuật toán được cài đặt đệ quy:
  - Rất khó thuật toán được gọi đệ quy bao nhiều lần
  - Cần phải chuyển thuật toán về dạng công thức đệ quy tổng quát
- Trình tự đánh giá
  - Xây dựng công thức đệ quy tổng quát (từ mô tả của thuật toán)
  - Dùng các phương pháp giải công thức đệ quy tổng quát đưa ra dạng O-lớn
    - Phương pháp phân rã
    - Phương pháp thế
    - Cây đệ quy
    - Định lý thợ

Thuật toán đệ quy

- Xây dựng công thức đệ quy tổng quát
  - Xác định trường hợp cơ sở:
    - Độ phức tạp tính toán trong trường hợp cơ sở
    - Với thuật toán đầu vào tối thiểu là 0 (không có đầu vào âm)
  - Xác định trường hợp tổng quát:
    - Lời gọi đệ quy thực hiện bao nhiều lần
    - Đâu là trường hợp sẽ dẫn đến thời gian tồi nhất
    - Chi phí phụ bên cạnh lời gọi đệ quy là bao nhiêu

```
int sum_rec(int *A, int n)
{
    if (n <= 0) return 0;
    return A[n - 1] + sum_rec(A, n - 1);
}
```

Trường hợp cơ sở n=0, thời gian O(1)
Trường hợp tổng quát với n>0: T(n) = T(n-1) + O(1)

$$T(n) = \begin{cases} 1, n = 0 \\ T(n-1) + 1, n > 0 \end{cases}$$

Thuật toán đệ quy

```
int binSearch_rec(int *A, int start, int end, int key)
{
    if (start > end) return -1;
    int mid = (start + end) / 2;
    if (A[mid] == key) return mid;
    if (A[mid] > key) return binSearch_rec(A,start,mid - 1,key);
    return binSearch_rec(A, mid + 1, end, key);
}
```

- Số lượng phần tử n = end – start + 1
- Trường hợp cơ sở: Dãy rỗng hoặc có 1 phần tử, thời gian O(1) hoặc 1
- Trường hợp tổng quát:
  - Có 3 lệnh return
  - 2 lệnh return ứng với lời gọi đệ quy sẽ cho thời gian thực hiện lâu nhất
- Lời gọi đệ quy
  - Kích thước bài toán con còn là n/2
  - Số lời gọi đệ quy : 1 (hoặc với if hoặc với else)
  - Chi phí phụ là hằng số, O(1)

$$T(n) = \begin{cases} 1, n = 0 \\ T\left(\frac{n}{2}\right) + 1, n > 0 \end{cases}$$

Thường khi đánh giá theo O-lớn ta chỉ cần quan tâm tới trường hợp tổng quát, nên có thể viết gọn lại

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Thuật toán đệ quy

Hàm MergeSort
- Gọi đệ quy 2 lần
- Kích thước bài toán con là n/2
- Chi phí phụ là O(n) + O(1) = O(n)

O(n) có thể biểu diễn bằng n cho gọn vì nó sẽ ko làm thay đổi tốc độ tăng trong O-lớn

```
MergeSort(int A[], int start, int end)
{
   if(start<end)
   {
          int mid = (start+end)/2;
          MergeSort(A, start, mid);
          MergeSort(A, mid+1, end);
          Merge(A,start,mid,end);
   }
}
```

Hàm Merge trộn 2 danh sách với thời gian O(n)

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Thuật toán đệ quy

- **Phương pháp phân rã**

$$T(n) = \begin{cases} 1, n = 0 \\ T(n-1) + 1, n > 0 \end{cases}$$

$$T(n) = T(n-1) + 1 = T(n-2) + 1 + 1 = T(n-3) + 1 + 1 + 1 = T(1) + 1 + .. + 1 = 1 + 1 + .. + 1 = n$$

Thời gian cỡ $O(n)$
Phương pháp này chủ yếu cho các công thức đệ quy dạng đơn giản

Thuật toán đệ quy

Ví dụ: Hãy giải các công thức đệ quy sau và đưa ra dạng O-lớn

a) T(n) = T(n/2) + 1
b) T(n) = 2T(n/3) + 1
c) T(n) = T(n-1) + 2
d) T(n) = 3T(n-1)+1

Thuật toán đệ quy

- **Phương pháp thay thế:**
  - Dự đoán dạng của O-lớn (cần có kinh nghiệm)
  - Chứng minh dự đoán bằng quy nạp
- Chú ý:
  - Trong O-lớn việc sai khác 1 KHÔNG ảnh hưởng đến tốc độ tăng của hàm, nên $T(n) = 2T(\lfloor n/2 \rfloor) + n$, $T(n) = 2T(\lceil n/2 \rceil) + n$ và $T(n) = 2T(n/2) + n$ là có tốc độ tăng giống nhau
  - Việc cộng thêm số hạng tăng chậm hơn KHÔNG ảnh hưởng tới tốc độ tăng, nên $T(n) = T(n/2) + n$ và $T(n) = T(n/2) + 100n + 5$ có tốc độ tăng giống nhau

Phương pháp thay thế

- Xác định cận trên của công thức đệ quy

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Dự đoán $T(n) = O(n\log n)$
Cần chứng minh $T(n) \leq cn\log n$ với hằng số $n > 0$ được chọn phù hợp

- Giả sử $T(n) \leq cn\log n$ đúng với $n/2$ tức là $T(\frac{n}{2}) \leq c\frac{n}{2}\log\frac{n}{2}$.
- Thay vào $T(n)$

$$T(n) \leq 2\left(c\frac{n}{2}\log\frac{n}{2}\right) + n \quad \leq cn\log\frac{n}{2} + n = cn\log n - cn\log 2 + n \quad \leq cn\log n$$

Đúng với $c \geq 1$
Ta cần chỉ ra kết quả quy nạp này đúng trong mọi trường hợp (đúng cả trong trường hợp cơ sở).

Phương pháp thay thế

- Giả sử trường hợp cơ sở $T(1) = 1$ nhưng $c1\log 1 = 0$. Kết quả quy nạp sai trong trường hợp cơ sở.
  - Ta có thể giải quyết vấn đề này bằng cách chọn giá trị n_0 sao cho $T(n) = cn\log n$ với $n \geq n_0$.

VD với $n = 2$ thì $T(2) = 2T(1) + 2 = 4 < c2\log 2$
ta chọn hằng số c đủ lớn đủ lớn (VD $c = 5$).

- Vậy $T(n) = O(n\log n)$ với $c = 5$ và $n \geq 2$
- Nhận xét:
  - Phương pháp thay thế chặt trẽ về mặt toán học
  - Yêu cầu phải dự đoán được dạng → ta co thể dự đoán dạng tốc độ tăng lớn rồi giảm dần để tìm dự đoán sát nhất có thể

Phương pháp thay thế

Ví dụ. Hãy chứng minh

a) $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$ là $O(\log n)$

b) $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ là $O(n \log n)$

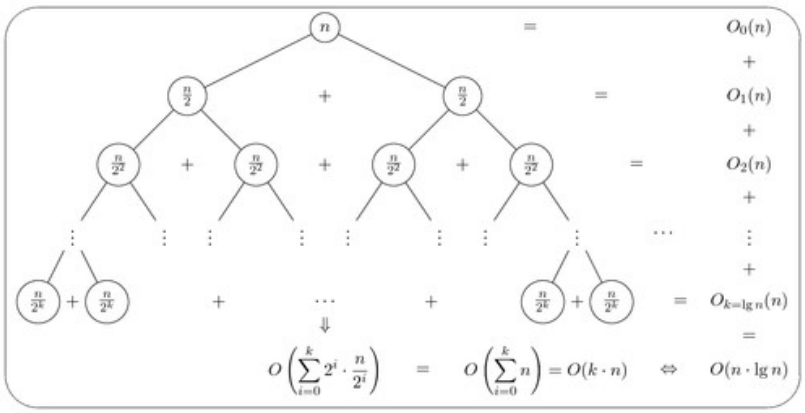
c) $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor + 12\right) + 1$ là $O(\log n)$

d) $T(n) = 4T\left(\frac{n}{2}\right) + n$ là $O(n^2)$

e) $T(n) = 2T\left(\frac{n}{2}\right) + 1$ là $O(n)$

Thuật toán đệ quy

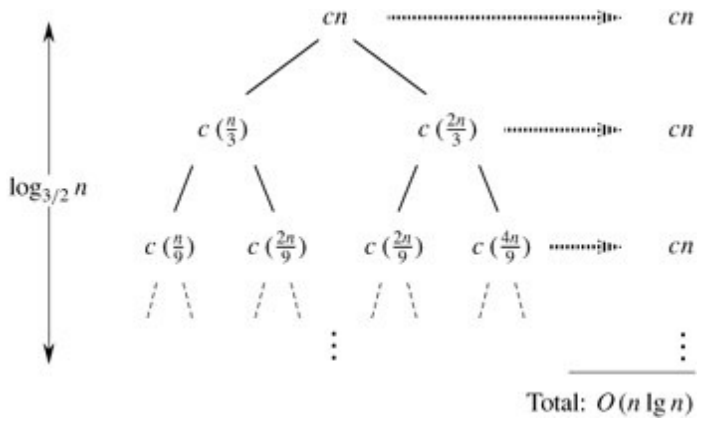
- **Cây đệ quy**:
  - Tìm dạng O-lớn một cách trực quan, nhưng chưa chặt trẽ về toán học
  - Có thể chứng minh lại bằng phương pháp thế



- Cây đệ quy cho mergeSort $T(n) = \begin{cases} O(1) & nếu\ n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & nếu\ n > 1 \end{cases}$

Cây đệ quy

- Công thức đệ quy
  $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$
- Cây đệ quy lệch
  - Nhánh đi theo $n/3$ là thấp nhất
  - Nhánh đi theo $2n/3$ là cao nhất



Cây đệ quy

- Dùng phương pháp thế để chứng minh lời giải công thức đệ quy tìm được.
  VD. $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) = O(n\log n)$
- $T(n) \le T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$
  $\le d\frac{n}{3}\log\frac{n}{3} + d\frac{2n}{3}\log\frac{2n}{3} + cn$
  $= \frac{dn}{3}\log n - \frac{dn}{3}\log 3 + \frac{2dn}{3}\log 2n - \frac{2dn}{3}\log 3 + cn$
  $= dn\log n - dn\log 3 + \frac{2dn}{3}\log 2 + cn$
  $\le dn\log n$

  Với $d \ge \frac{c}{\log 3 - \frac{2}{3}}$ (chú ý $\log 2 = 1$)

Cây đệ quy

- **VD 1.** Xác định một cận trên tốt cho công thức đệ quy $T(n) = 3T\left(\frac{n}{2}\right) + n$ dùng phương pháp thế để xác nhận lại kết quả.
- **VD 2.** Vẽ cây đệ quy cho $T(n) = 4T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + cn$ với c là hằng số. Đưa ra tiệm cận chặt cho công thức đệ quy trên. Xác nhận lại lời giải bằng phương pháp thế.

Thuật toán đệ quy

- Định lý thợ - Master theorem
  - Dùng để giải các công thức đệ quy dạng $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, với a $\geq 1, b > 1, f(n)$ là hàm tiệm cận dương
  - Bài toán ban đầu được chia thành $\boldsymbol{a}$ bài toán con có kích thước mỗi bài là $\boldsymbol{n/b}$, chi phí để tổng hợp các bài toán con là $\boldsymbol{f(n)}$

    VD. Thuật toán sắp xếp trộn
    chia thành 2 bài toán con, kích thước $n/2$. Chi phí tổng hợp 2 bài toán con là $O(n)$
  - Với các dạng công thức đệ quy không áp dụng được định lý thợ, vẫn có thể áp dụng các phương pháp trước để giải

Định lý thợ

Định lý thợ xét mối quan hệ về tốc độ tăng của $n^{\log_b a}$ và $f(n)$

- TH 1. Nếu $f(n) = O(n^{\log_b a - \epsilon})$, với hằng $\epsilon > 0$ thì $T(n) = \Theta(n^{\log_b a})$
- TH 2. Nếu $f(n) = \Theta(n^{\log_b a})$ thì $T(n) = \Theta(n^{\log_b a} \log n)$
- TH 3. Nếu $f(n) = \Omega(n^{\log_b a + \epsilon})$, với hằng $\epsilon > 0$, và nếu $af\left(\frac{n}{b}\right) < cf(n)$ với hằng $c < 1$ và với mọi n đủ lớn thì $T(n) = \Theta(f(n))$

- Có thể hiểu dạng
  - TH 1. tốc độ tăng $n^{\log_b a}$ nhanh hơn so với $f(n)$ $(\lim_{n\to\infty} \frac{n^{\log_b a}}{f(n)} = \infty)$
  - TH 2. tốc độ tăng $n^{\log_b a}$ bằng với $f(n)$ $(0 < \lim_{n\to\infty} \frac{n^{\log_b a}}{f(n)} = C < \infty)$
  - TH 3. tốc độ tăng $n^{\log_b a}$ chậm hơn so với $f(n)$

Dùng định lý thợ

Áp dụng định lý thợ:

- $T(n) = 9T\left(\frac{n}{3}\right) + n.$
  $a = 9, b = 3$ và $f(n) = n$ ta có $n^{\log_b a} \equiv n^{\log_3} = n^2$. Đây là trường hợp 1 (với $\epsilon = 1$) do đó $T(n) = \Theta(n^2)$
- $T(n) = T\left(\frac{2n}{3}\right) + 1.$
  $a = 1, b = 3/2$ và $f(n) = 1$ ta có $n^{\log_{3/2} 1} = n^0 = 1$. Đây là trường hợp 2, do đó $T(n) = \Theta(\log n)$
- $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$
  $a = 3, b = 4$ và $f(n) = n \log n$ ta có $n^{\log_b a} \equiv n^{\log_4 3} = O(n^{0.793})$, $f(n) = \Omega(n^{\log_4 3 + \epsilon})$ với $\epsilon \approx 0.2$, $af(n/b) < cf(n) \equiv 3f\left(\frac{n}{4}\right) < cn \log n$ với $c = \frac{3}{4}$ do vậy $T(n) = \Theta(n \log n)$ (TH3)

Dùng định lý thợ

- **<u>Chú ý</u>**: Không phải trường hợp nào cũng áp dụng được định lý thợ !
- VD. $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$
  $a = 2, b = 2$ và $f(n) = n \log n$
  $n^{\log_b a} \equiv n^{\log_2 2} = n$ do đó có vẻ áp dụng trường hợp 3. Tuy nhiên $f(n) = n \log n$ tiệm cận lớn hơn $2f\left(\frac{n}{2}\right)$ với mọi hằng số ϵ do đó không thể áp dụng được.

Định lý thợ mở rộng

- Cho dạng hàm $T(n) = \begin{cases} c, n \leq 1 \\ aT(n-b) + f(n), n > 1 \end{cases}$
  - Với các hằng số $c, a > 0, b > 0, k \geq 0$ và hàm $f(n)$, nếu hàm $f(n)$có cận trên là $O(n^k)$ thì

$$T(n) = \begin{cases} O(n^k), a < 1 \\ O(n^{k+1}), a = 1 \\ O(n^k a^{\frac{n}{b}}), a > 1 \end{cases}$$

- Dạng hàm $T(n) = T(\alpha n) + T((1-\alpha)n) + \beta n$, trong đó $0 < \alpha < 1, \beta > 0$ thì $T(n) = O(nlogn)$

Dùng định lý thợ

Bài 1. Dùng định lý thợ để đưa ra các tiệm cận chặt cho các công thức đệ quy sau

- $T(n) = 4T\left(\frac{n}{2}\right) + n$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^2$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

Bài 2. Dùng định lý thợ để chứng minh thời gian thực hiện của thuật toán tìm kiếm nhị phân $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$ là $\Theta(\log n)$

Bài 3. Định lý thợ có thể áp dụng cho công thức đệ quy $T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log n$ được không, Tại sao? Tìm tiệm cận giới hạn trên cho $T(n)$

Dùng định lý thợ

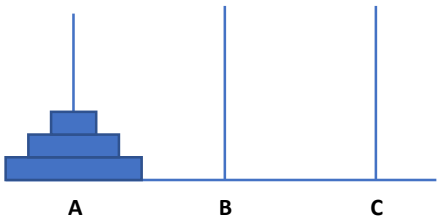
Bài 4. Đánh giá thời gian thực hiện của thuật toán đệ quy sau theo mô hình O-lớn

```
int findMin(int S[], int start, int end)
{
        if(start>=end) return S[end];

        int div = (end-start)/2;
        int A,B;

        A=findMin(S,start,start+div);
        B=findMin(S,start+div+1,end);

        return Min(A,B);
}
```

Trong đó hàm Min(A,B) trả về giá trị nhỏ nhất trong 2 số A, B và thời gian thực hiện là $\Theta(1)$

Thuật toán đệ quy

- **Bài toán tháp Hà Nội**
  - Có n đĩa với kích thước khác nhau và 3 cọc A, B, C
  - Ban đầu n đĩa nằm ở cọc A theo thứ tự đĩa nhỏ nằm trên và đĩa lớn nằm dưới
  - Tìm cách chuyển n đĩa này từ cọc A sang cọc B, sử dụng cọc C làm trung gian theo nguyên tắc
    - Mỗi lần chỉ được chuyển 1 đĩa trên cùng từ 1 cọc sang cọc khác
    - Không được phép để xảy ra tình trạng đĩa to nằm bên trên đĩa nhỏ



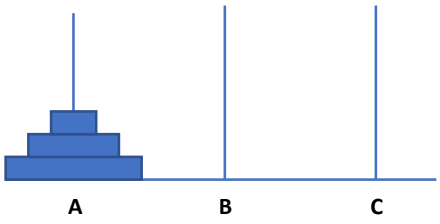
Lời giải với n=3

- B1: A → B
- B2: A → C
- B3: B → C
- B4: A → B
- B5: C → A
- B6: C → B
- B7: A → B

THÁP HÀ NỘI

Hàm move di chuyển từ cột A sang cột B với cột trung gian là C

```
void move(int n, char A, char B, char C)
{
   if (n == 1) {
      printf("Move 1 disk from %c to %c", A, B);
   }
   else {
      move(n - 1, A, C, B);
      move(1, A, B, C);
      move(n - 1, C, B, A);
   }
}
int main() {
   int n = 3;
   move(n, 'A', 'B', 'C’);
   return  0;
}
```



Lời giải với n=3

- B1: A → B
- B2: A → C
- B3: B → C
- B4: A → B
- B5: C → A
- B6: C → B
- B7: A → B

Thuật toán đệ quy

- Tính giá trị x^n nhanh (với x là giá trị thực và n là mũ nguyên dương)

```
double fastPower1(double x, int n)
{
   if (n == 0) return 1;
   if (n % 2 == 0) return fastPower(x, n / 2)*fastPower(x, n / 2);
   return fastPower(x, n / 2)*fastPower(x, n / 2)*x;
}
```

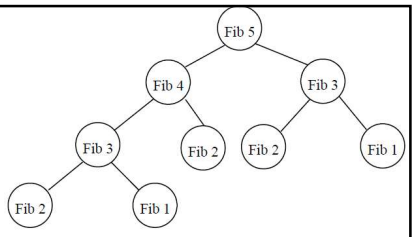
$$T_1(n) = 2T\left(\frac{n}{2}\right) + 1 = O(n)$$

Chỉ cần dùng thêm 1 biến phụ hạn chế bớt gọi đệ quy, hiệu quả thuật toán thay đổi rõ rệt

$$T_2(n) = T\left(\frac{n}{2}\right) + 1 = O(\log n)$$

```
double fastPower2(double x, int n)
{
   if (n == 0) return 1;
   double y = fastPower(x, n / 2);
   if (n % 2 == 0) return y*y;
   return y*y*x;
}
```

Thuật toán đệ quy



- Tính số Fibonacci

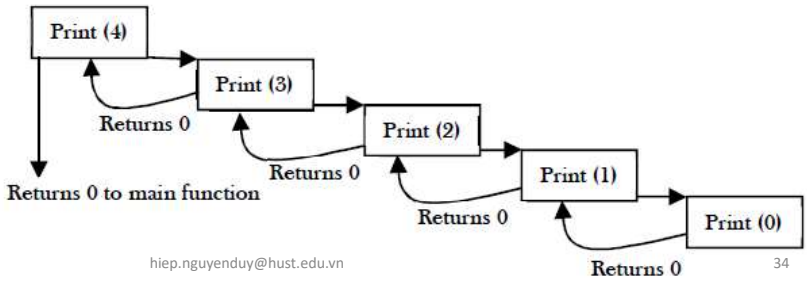
$$f(n) = \begin{cases} 1 & nếu\ n = 0,1 \\ f(n-1) + f(n-2) & nếu\ n \geq 2 \end{cases}$$

Tính $f(5)$

- $f(5) = f(4) + f(3) = f(3) + f(2) + f(3) = \cdots$
- **Đệ quy có nhớ**: Trường hợp bài toán con có thể trùng lặp
  - Để cải thiện hiệu năng của thuật toán được cài đặt đệ quy, nên hạn chế tối thiểu gọi đệ quy nếu có thể (thay bằng biến phụ, hoặc bảng tra nếu các bài toán con có xu hướng lặp lại)
  - Khi gặp bài toán con, kiểm tra xem lời giải đã có trong bảng tra hay chưa.
    - Nếu chưa có thì giải và cập nhật lời giải vào bảng tra
    - Nếu đã có thì lấy lời giải để sử dụng

Thuật toán đệ quy

```
public int Print(int n) {
   if (n == 0) // this is the terminating base case
      return 0;
   else {
      System.out.println(n);
      return Print(n - 1); // recursive call to itself again
   }
}
```



Thuật toán đệ quy

- Đệ quy trực tiếp VS Đệ quy gián tiếp

```
function F()
{
      if base condition satisfied
            then return some value(s)
      else
      {
         :
         call F();
         :
      }
}
```

```
function A()
{
      :
      call to B;
      :
}

function B()
{
      :
      call to A;
      :
}
```


Thuật toán quay lui
Back-tracking algorithm

Thuật toán quay lui

- Thuật toán quay lui
  - Là một cải tiến của lớp các bài toán vét cạn – brute force
  - Tại một thời điểm chỉ thử đi theo theo một hướng chứ không thử đồng thời tất cả các khả năng (dùng ít bộ nhớ phụ hơn)
  - Nếu hướng hiện tại gặp bế tắc – dead end, quay lại và thử đi theo hướng mới
  - Từng bước xây dựng lời giải bộ phận để thu được lời giải cuối cùng
  - Trong trường hợp tồi nhất thuật toán vẫn phải vét cạn hết tất cả các khả năng có thể có trong không gian các phương án tiềm năng
  - Để chứng minh bài toán không có lời giải, cần phải vét hết toàn bộ không gian tìm kiếm
  - Nếu không gian bài toán lớn, thời gian tìm lời giải có thể rất lâu
  - Thường cài đặt đơn giản dùng đệ quy, thay vì dùng vòng lặp (vẫn có thể cài đặt bằng vòng lặp được)

Thuật toán quay lui

- Một số bài toán dùng thuật toán quay lui
  - Bài toán tìm đường đi trong mê cung
  - Bài toán sinh tất cả chuỗi nhị phân độ dài n
  - Bài toán xếp hậu, mã đi tuần,…
  - Bài toán cái túi
  - Bài toán chu trình Hamiltonian
  - Bài toán tô màu đồ thị

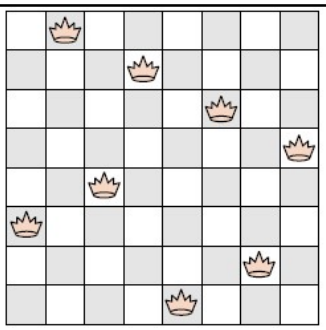
Thuật toán quay lui

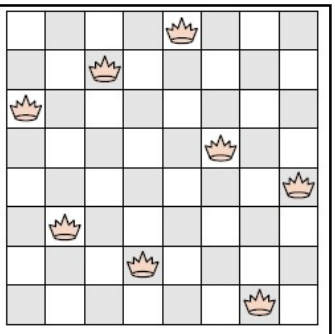
Sơ đồ tổng quát của thuật toán quay lui

```
Solve_from (Current_config) {
if (Current_config đã chứa lời giải đầy đủ)
        print Current_config
else
  Với tập các lựa chọn chưa được xét đến trong Current_config
    {
      Thêm 1 lựa chọn P;
      Cập nhật lại Current_config;
      solve_from(Current_config);
      Loại bỏ lựa chọn P khỏi Current_config; //quay lui
    }
}
```


Thuật toán quay lui

- **Bài toán 8 con hậu**: "Hãy xếp 8 con hậu trên bàn cờ 8x8 sao cho chúng không thể ăn lẫn nhau"



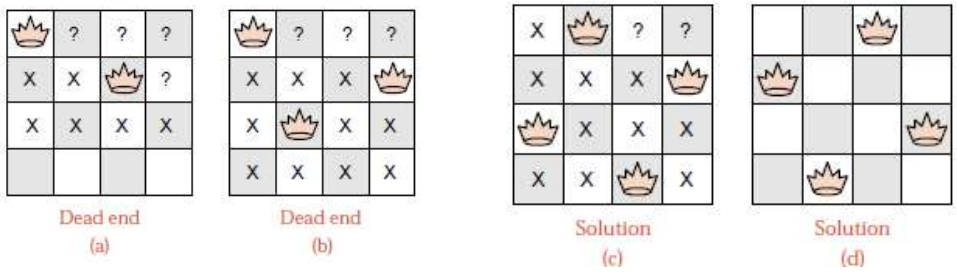


Ý tưởng:

- Các quân hậu cần được xếp riêng trên mỗi cột (hoặc hàng)
- Thử xếp các quân hậu trong các cột lần lượt từ 1 tới 8
- Mỗi cột thử đặt quân hậu tại lần lượt các hàng (của cột đó)
- Nếu có vị trí đặt hợp lệ thì chuyển qua đặt hậu ở cột kế bên
- Ngược lại, nếu không thể tìm được vị trí đặt hậu tại cột hiện tại, cần quay lui lại lựa chọn vị trí đặt của cột ngay trước để sửa lại
- Nếu đặt đủ 8 hậu → tìm được 1 phương án giải

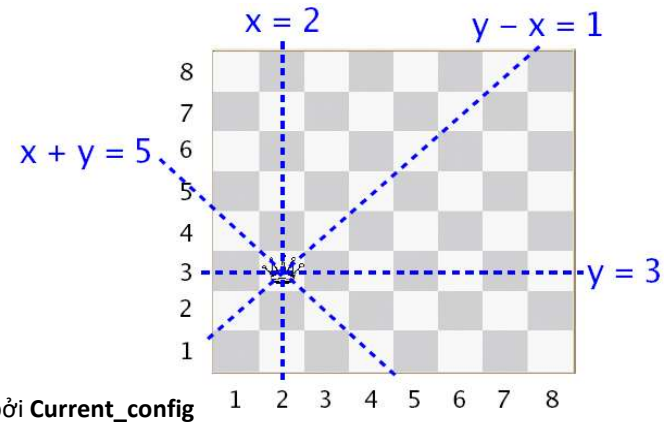
Thuật toán quay lui

- **Dead end**: trạng thái chưa kết thúc, nhưng ta không thể đặt thêm được 1 quân hậu nào nữa.
- Khi rơi vào trạng thái **dead end** ta phải tiến hành quay lui (back track) lại lựa chọn gần nhất để thử một khả năng có thể khác.



Dead end với bài toán xếp 4 hậu

Kiểm tra An toàn



solve_from (**Current_config**){
if Current_config đã chứa đủ 8 hậu
print **Current_config**
else
Với tập p các ô trên bàn cờ mà chưa bị ảnh hưởng bởi **Current_config**
{
Thêm 1 quân hậu vào p;
Cập nhật lại **Current_config**
solve_from(**Current_config**);
Loại bỏ quân hậu khỏi p của **Current_config**;
}
}

Giải thuật

```
function Try (column) {
  for (row = 1; row <= 8; row++) {
      if (Check [row, column] là an toàn) {
          Đặt con hậu vào vị trí [row, column];
          if (column == 8)
              In kết quả;
          else
              Try (column + 1);
          Xóa con hậu khỏi vị trí [row, column];
      }
  }
}
```

Thử lần lượt từng vị trí hàng

Nếu vị trí thử không bị con hậu nào tấn công

Con hậu thứ 8 là an toàn

Gọi đệ quy với con hậu tiếp

Xóa để tiếp tục thử vị trí [row+1, column]

Thuật toán quay lui

- Bài toán sinh các chuỗi nhị phân độ dài n (tổng quát: sinh các hoán vị của tập n phần tử)

```
void printSolution(int *x, int n) {
   for (int k = 0; k < n; k++)
      printf("%d", x[k]);
   printf("\n");
}
int TRY(int *x, int n, int k) {
   for (int v = 0; v <= 1; v++) {
      x[k] = v;
      if (k == n - 1) printSolution(x,n);
      else TRY(x, n, k + 1);
   }
}
int main() {
   int x[5],n=5;
   TRY(x,n,0);
}
```

```
import java.util.*;
public class BinaryStrings {
int[] A;

    public BinaryStrings(int n) {
       A = new int[n];
    }

    public void binary(int n) {
        if (n <= 0) {
           System.out.println(Arrays.toString(A));
        }
        else {
            A[n - 1] = 0;
            binary(n - 1);
            A[n - 1] = 1;
            binary(n - 1);
        }
    }

    public static void main(String[] args) {
        int n = 4;
        BinaryStrings i = new BinaryStrings(n);
        i.binary(n);
    }
}
```

Thuật toán quay lui

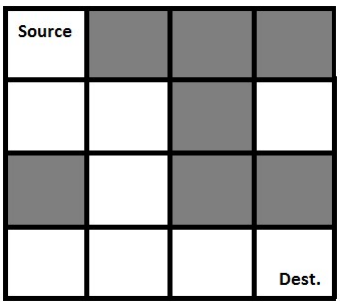
- Bài toán sinh các chuỗi nhị phân mở rộng: Không có 2 bit 1 đứng cạnh nhau
- Việc check điều kiện được gộp vào bước continue trong vòng lặp vì bài toán khá đơn giản

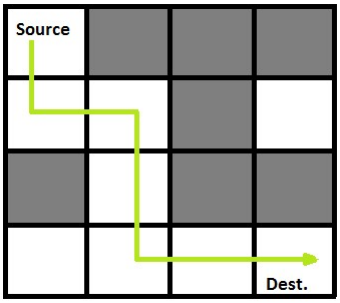
```
void printSolution(int *x, int n) {
   for (int k = 0; k < n; k++)
   printf("%d", x[k]);
   printf("\n");
}

int TRY(int *x, int n, int k) {
   for (int v = 0; v <= 1; v++) {
      if (k>0 && v + x[k - 1] == 2) continue;
      x[k] = v;
      if (k == n - 1) printSolution(x, n);
      else TRY(x, n, k + 1);
   }
}

int main() {
   int x[5], n = 5;
   TRY(x, n, 0);
}
```


Thuật toán quay lui

- Bài toán tìm đường đi trong mê cung
  - Mê cung có thể biểu diễn thành ma trận nhị phân Tường sẽ là bit 0, điểm bắt đầu [0,0], kết thúc [N-1,M-1]

```
{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
{1, 1, 1, 1}
```






```
void printSolution(int sol[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
               printf(" %d ", sol[i][j]);
        printf("\n");
    }
}
```

```
bool isSafe(int maze[N][N], int x, int y)
{
    if (x >= 0 && x < N && y >= 0
        && y < N && maze[x][y] == 1)
    return true;

    return false;
}
```

```
bool solveMazeUtil(int maze[N][N], int x,int y, int sol[N][N])
{
    if (x == N - 1 && y == N – 1 && maze[x][y] == 1) {
        sol[x][y] = 1;
        return true;
    }

    if (isSafe(maze, x, y) == true) {
        // Check if the current block is already part of solution path.
        if (sol[x][y] == 1) return false;

        sol[x][y] = 1; // mark x, y as part of solution path

        if (solveMazeUtil(maze, x + 1, y, sol)== true) return true;
        if (solveMazeUtil(maze, x, y + 1, sol)== true) return true;
        if (solveMazeUtil(maze, x-1, y, sol)== true) return true;
        if (solveMazeUtil(maze, x, y - 1, sol)== true) return true;

        sol[x][y] = 0; /* unmark x, y as part of solution path */
        return false;
    }

    return false;
}
```

$$T(n) = O(2^{NM})$$

https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/

Thuật toán quay lui

- Bài toán giải ô chữ SUDOKU: 2x2, 3x3, 4x4

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

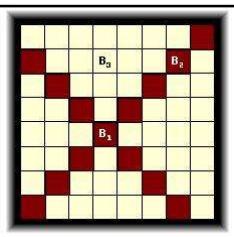
$$T(n) = O(9^{n^2})$$

https://www.geeksforgeeks.org/sudoku-backtracking-7/

```
Input:
grid = { {3, 0, 6, 5, 0, 8, 4, 0, 0},
         {5, 2, 0, 0, 0, 0, 0, 0, 0},
         {0, 8, 7, 0, 0, 0, 0, 3, 1},
         {0, 0, 3, 0, 1, 0, 0, 8, 0},
         {9, 0, 0, 8, 6, 3, 0, 0, 5},
         {0, 5, 0, 0, 9, 0, 6, 0, 0},
         {1, 3, 0, 0, 0, 0, 2, 5, 0},
         {0, 0, 0, 0, 0, 0, 0, 7, 4},
         {0, 0, 5, 2, 0, 6, 3, 0, 0} }
Output:
          3 1 6 5 7 8 4 9 2
          5 2 9 1 3 4 7 6 8
          4 8 7 6 2 9 5 3 1
          2 6 3 4 1 5 9 8 7
          9 7 4 8 6 3 1 2 5
          8 5 1 7 9 2 6 4 3
          1 3 8 9 4 7 2 5 6
          6 9 2 3 5 1 8 7 4
          7 4 5 2 8 6 3 1 9
```

Thuật toán quay lui

```
5 8
1 0 0 1 1 1 1 1
0 1 1 1 0 0 1 1
1 0 1 1 1 1 0 1
1 0 1 0 1 1 1 1
1 0 1 1 1 1 0 1
```



```
  SEND
+ MORE
 MONEY
```

- Bài 1. Liệt kê các hoán vị của xâu "ABCDE" và "AABEB"
- Bài 2. In ra các xâu độ dài k với các ký tự lấy ra từ xâu "ABCDEFGH"
- Bài 3. Điền các chữ số 0-9 vào thay cho các ký tự trong đoạn mật mã sau
- Bài 4. Xây dựng thuật toán in ra những cách xếp k quân tượng trên bàn cờ vua kích thước nxn sao cho các quân tượng này không đe dọa nhau
- Bài 5. Cho đầu vào là file văn bản gồm
  - dòng đầu tiền là số hàng và số cột
  - Các dòng tiếp theo là các dòng chứa giá trị 1 và 0 tương ứng

  Giả sử xuất phát từ 1 vị trí số 1 ngẫu nhiên nào đó, và chỉ có thể đi theo 4 hướng (lên, xuống, trái, phải) theo các vị trí số 1 (vị trí 0 là vị trí không thể đi được). Hãy in ra quãng đường dài nhất mà có thể đi từ vị trí ban đầu.

Thuật toán quay lui

- Bài 6. Cho đầu vào là 1 danh sách số nguyên gồm n phần tử với thứ tự bất kỳ. Hãy xây dựng chương trình in ra dãy số sao cho phần tử có giá trị lớn nhất ở giữa, sau đó giá trị các phần tử giảm dần về 2 phía (có thể có nhiều cách in thỏa mãn)
  VD. 54,7,25,45,78,12,6,8 → 7, 12, 54, 78, 45, 25, 8, 6
- Bài 7. Bài toán cắt dây
  Anh công nhân A được giao n cuộn dây với độ dài d bằng nhau (d<=50). Trong quá trình làm việc, anh cắt các dây này thành các đoạn ngắn hơn theo 1 cách ngẫu nhiên nào đó (anh cũng không nhớ được). Tuy nhiên vì chất lượng các dây không tốt nên anh muốn trả lại. Kho đồng ý nhập lại, nhưng cần trả lại dây theo độ dài như ban đầu (anh cũng quên mất độ dài ban đầu). Hãy xây dựng thuật toán tìm kích thước ban đầu ngắn nhất và số lượng dây n

  5 2 1 5 2 1 5 2 1 → d min = 6
  1 2 3 4 → d min =5

Thuật toán quay lui

12
45
56
70
120
45
56
110
65
45
57
70
68

Bài 8. Tổ chức thi kéo co

Để tổ chức thi kéo co cho 1 nhóm n người tham gia team building, bên tổ chức sự kiện quyết định chia nhóm thành 2 đội, và để đảm bảo công bằng, họ chia nhóm sao cho

- Tổng số lượng người mỗi nhóm chênh lệch nhau không quá 1
- Tổng cân nặng của mỗi nhóm chênh lệch nhau là nhỏ nhất

Bạn nhận được đầu vào là danh sách người và cân nặng mỗi người, hãy xây dựng thuật toán để chia nhóm đảm bảo yêu cầu trên

File đầu vào

- Dòng đầu là tổng số người (n<100)
- Các dòng tiếp theo là cân nặng của từng người (<=200kg)

Thuật toán quay lui

8 12
1 2
1 6
1 8
2 3
2 6
3 4
3 5
4 5
4 7
5 6
6 7
6 8

Bài 9. Một công ty muốn xây dựng mạng lưới nhận hàng và giao hàng trên địa bàn thành phố. Mục tiêu hiện tại là:

- điểm giao dịch phải đặt mỗi khu phố X,
- hoặc ngay khu phố lân cận của X.

Cho đầu vào là danh sách các khu phố lân cận nhau, hãy xây dựng thuật toán đưa ra cách đặt mạng lưới điểm giao dịch sao cho số điểm cần đặt là ít nhất mà vẫn thỏa mãn mục tiêu

File dữ liệu

- Đầu vào là file text với dòng đầu là số khu phố và số liên kết giữa các khu phố
- Các dòng tiếp theo là liên kết giữa các khu phố (coi tên khu phố là dạng số

Với đầu vào ở trên thì số điểm đặt nhỏ nhất là 2