ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN

Bài 4

Phương pháp chia để trị
(Divide and conquer)

Nguyễn Thị Hồng Minh
minhnth@gmail.com

Nội dung

1. **Giới thiệu**
2. **Mô hình và lược đồ giải thuật chia để trị**
3. **Một số vấn đề chú ý khi thiết kế giải thuật chia để trị**
4. **Ví dụ**

Giới thiệu

- **Ý tưởng giải thuật chia để trị**:

  Là phương pháp thiết kế thuật toán dựa trên 2 thao tác chính:

  - Chia (*divide*): phân rã bài toán ban đầu thành các bài toán con có kích thước nhỏ hơn, có cùng cách giải.
  - Trị (*conque*): giải từng bài toán con (theo cách tương tự bài toán đầu - đệ qui) rồi tổng hợp các lời giải để nhận kết quả của bài toán ban đầu.

  Việc "Phân rã": thực hiện trên miền dữ liệu (chia miền dữ liệu thành các miền nhỏ hơn tương đương 1 bài toán con) đến lúc miền dữ liệu đủ nhỏ để bài toán có thể giải xác định (trường hợp suy biến trong đệ qui)

Mô hình và lược đồ giải thuật

- **Mô hình**

Xét bài toán P trên miền dữ liệu R.

Gọi D&C(R) là thuật giải P trên miền dữ liệu R.

Nếu R có thể phân rã thành n miền con ($R= R_1 \square R_2 \square \ldots \square R_n$)

Với R_0 là miền đủ nhỏ để D&C(R_0) có lời giải thì lược đồ giải thuật chia để trị:

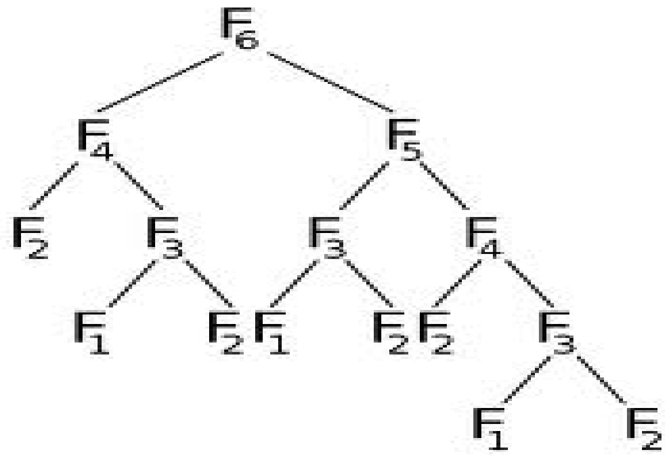
```
D&C(R)≡
|    if(R=R_0)
|        Giải D&C(R_0);
|    else
|        Chia miền R thành R_1,R_2,…,R_n
|        for(i=1…n)
|             D&C(R_i)
|        Tổng hợp để nhận lời giải.
End.
```

Mô hình và lược đồ giải thuật

- **Tính đúng của giải thuật chia để trị**
  - Chia để trị sử dụng kĩ thuật đệ qui, thông thường là đệ qui nhiều nhánh
  - Tính đúng của thuật toán D&C có thể được chứng minh như đối với giải thuật đệ qui sử dụng qui nạp.
  - Qui nạp
    - Qui nạp theo kích thước miền dữ liệu (kích thước dữ liệu bài toán)
    - Cơ sở qui nạp: Giải thuật đúng trong trường hợp suy biến R_0
    - Giả thiết qui nạp: Giả sử giải thuật đúng với miền dữ liệu R (n=|R|)
    - Tổng quát: Giải thuật đúng với miền dữ liệu R_1 có $|R_1| = n+1$
  - Nếu giải thuật đã được khử đệ qui thì sử dụng bất biến vòng lặp

Mô hình và lược đồ giải thuật

- **Một số chú ý khi thiết kế giải thuật chia để trị**
  - Lời gọi đệ qui trên nhiều bài toán, nhiều lần có thể dẫn tới hiện tượng tràn vùng nhớ đệm ⇒ khử đệ qui cho thuật toán D&C.
  - Cần có chiến lược phân rã hợp lý miền dữ liệu để thuật giải tốt nhất.
  - Hạn chế thường gặp của thuật toán D&C: Việc phân rã có thể dẫn tới một số bài toán con trùng nhau.
    - Ví dụ: Tìm số Fibonacci thứ 6



⇒ Phương pháp Qui hoạch động (Bottom-Up).

Ứng dụng

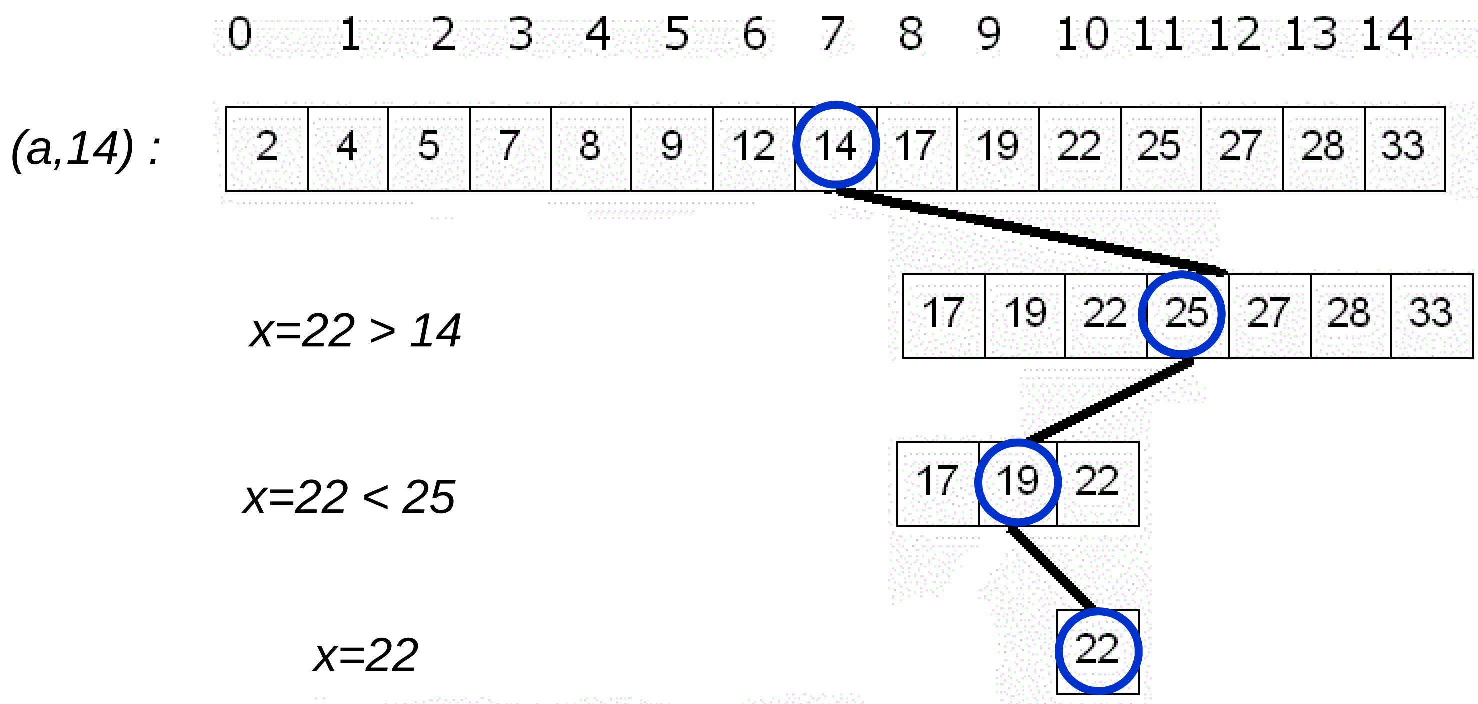
- **Một số bài toán giải bằng chia để trị**
  - Bài toán sắp xếp: thuật toán Quicksort, Mergesort
  - Bài toán tìm kiếm: thuật toán tìm kiếm nhị phân
  - Bài toán nhân số lớn, nhân ma trận kích thước lớn
  - Tính chuỗi Fourier
  - Bài toán phân tích ngữ nghĩa (xử lí ngôn ngữ)
  - …

Ví dụ 1: Tìm kiếm nhị phân

- **Bài toán: Tìm kiếm nhị phân trên mảng được sắp**
  - Cho mảng *n* phần tử được sắp theo thứ tự (tăng dần) và một giá trị *x* bất kỳ. Kiểm tra xem phần tử *x* có trong dãy không ?
  - Phân tích ý tưởng: so sánh giá trị *x* với phần tử giữa của dãy tìm kiếm. Dựa vào giá trị này sẽ quyết định giới hạn tìm kiếm ở bước kế tiếp là nửa trước hay nửa sau dãy.

Ví dụ 1: Tìm kiếm nhị phân

- **Tìm kiếm nhị phân trên mảng được sắp**



Ví dụ 1: Tìm kiếm nhị phân

- **Lược đồ thuật toán**

```
BinarySearch(a,x, L,R):int ≡
    //Tìm kiếm phần tử x trên dãy a từ vị trí L đến R
    if (L=R) return (x=aL ? L : -1)
    else
        M = (L+R)/2;
        if (x = aM) return (M);
        else
            if (x<aM) BinarySearch(a,x,L,M)
            else BinarySearch(a,x,M+1,R)
        endif;
    endif;
End.
```

Ví dụ 1: Tìm kiếm nhị phân

- **Lược đồ chương trình** có lời gọi `BinarySearch`

```
main() ≡
    Input (a,n);
    Input x;
    result = BinarySearch(a,x,0,n-1);
    Output (result);
End.
```

Ví dụ 1: Tìm kiếm nhị phân

- **Chứng minh tính đúng của thuật toán**

  Chứng minh qui nạp theo số phần tử của dãy

  - Cơ sở qui nạp: $n = R\text{-}L+1 = 1$ (dãy chỉ có 1 phần tử)
    - Câu lệnh `return (x=a`$_L$` ? L : -1)` trả lại giá trị L hoặc -1
  - Giả thiết qui nạp: Thuật toán đúng với mọi dãy có độ dài $n = R\text{-}L+1$
    - Tức là `BinarySearch(a,x, L,R)`trả về đúng kết quả tìm kiếm *x* với mọi dãy có độ dài $1 \leq n' \leq n = R\text{-}L+1$
  - Tổng quát: Chứng minh thuật toán đúng với $n+1 = R\text{-}L+2$
    - $M=(L+R+1)/2$; $L \leq M \leq R$
    - Nếu $x=a_M$ thì kết quả trả về là M : đúng
    - Nếu $x<a_M$ thì kết quả là của bài toán $x \in a_L..a_M$? Theo giả thiết qui nạp thì `BinarySearch(a,x,L,M)`đúng vì $1 \leq M\text{-}L+1=(R\text{-}L+1)/2+1 \leq R\text{-}L+1$
    - Nếu $x>a_M$ tương tự

Ví dụ 1: Tìm kiếm nhị phân

- **Độ phức tạp thuật toán**

$$T(n) = \begin{cases} 1 \quad \text{khi } n = 1 \\ T(n/2) + 1 \text{ khi n} > 1 \end{cases}$$

$\Rightarrow$ T(n) = O(log n)

- **Khử đệ qui thuật toán đưa về dạng lặp**

Sử dụng lược đồ đệ qui dạng ED[AP]

Ví dụ 2: Bài toán MinMax

- **Tìm phần tử lớn nhất, nhỏ nhất của dãy**
  - Bài toán: Cho mảng *n* phần tử, tìm giá trị lớn nhất và nhỏ nhất các phần tử trong dãy.
  - Phân tích ý tưởng:
    - Sử dụng phương pháp truyền thống: Hai vòng lặp độc lập tìm *max*, *min*.

      Độ phức tạp thuật toán O(*n*); Số phép so sánh 2(*n*-1)
    - Sử dụng phương pháp chia để trị:
      - Chia: Chia dãy thành 2 phần
      - Trị: Tìm giá trị *max*, *min* trong mỗi phần
      - Kết hợp: So sánh giá trị *min*, *max* đã tìm được

Ví dụ 2: Bài toán MinMax

- **Lược đồ thuật toán**

```
MinMax(a,L,R):(int,int) ≡
   //Tìm giá trị min,max dãy a từ vị trí L đến R
   if (R-L<=1){
      return(min(aL,aR),max(aL,aR));
   }
   else {
      (min1,max1) = MinMax(a,L,(L+R)/2);
      (min2,max2) = MinMax(a,(L+R)/2+1,R);
      return(min(min1,min2),max(max1,max2));
   }
End.
```

Lời gọi ban đầu với dãy *n* phần tử: `MinMax(a,0,n-1)`

Ví dụ 2: Bài toán MinMax

- **Tính đúng của thuật toán**
  - Chứng minh qui nạp theo số phần tử của dãy $n = R - L + 1$.
    - Cơ sở: n <=1
    - Giả thiết qui nạp: Thuật toán đúng với $n = R - L + 1$

      Tức là: `Minmax(a,L,R,Min,Max)` trả về giá trị *Min*, *Max* trong khoảng $L..R$ (từ $a_L,..,a_R$)
    - Tổng quát: Chứng minh thuật toán đúng với $n = R - L + 2$

Ví dụ 2: Bài toán MinMax

- **Tìm phần tử lớn nhất, nhỏ nhất của dãy**
  - Độ phức tạp thuật toán

$$T(n) = \begin{cases} 2 & \text{khi } n = 2 \\ 2T(n/2) + 2 & \text{khi n} > 1 \end{cases}$$

$\Rightarrow$ T(n) = O(n)

Không khác với thuật toán truyền thống, hai vòng lặp tìm *max*, *min* độc lập

  - Tuy nhiên, so sánh số phép so sánh: (Giả sử $n=2^m$)

$T(n) = T(2^m)=2T(2^{m-1})+2 = 2(2T(2^{m-2})+2) + 2 = 2^2T(2^{m-2})+2^2+2 =$

$=\ldots=2^{m-1}T(2^{m-m+1})+2^{m-2}+\ldots+2^2+2= 2^{m-1}+\ldots+2^2+2$

$= 2^m-1-1= n-2$

$\Rightarrow$ Số phép so sánh bằng khoảng ½ so với phương pháp truyền thống

Ví dụ 3: Cặp điểm gần nhất

- **Tìm kiếm cặp điểm gần nhất (closest-pair problem)**
  - Bài toán: Cho một tập *n* điểm trong mặt phẳng, tìm cặp điểm gần nhau nhất (theo nghĩa khoảng cách Euclit)

$$P = \{(x_{1,}y_1),(x_{2,}y_2),\dots,(x_{n,}y_n)\}$$

Tìm cặp điểm pj, pj sao cho $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ nhỏ nhất

  - Phân tích ý tưởng:
    - o Nếu dùng phương pháp Brute-force: So sánh khoảng cách mọi cặp điểm thì độ phức tạp là O(n^2)
    - o Chia để trị: Tìm giải thuật có độ phức tạp tốt hơn O(*n*log*n*)

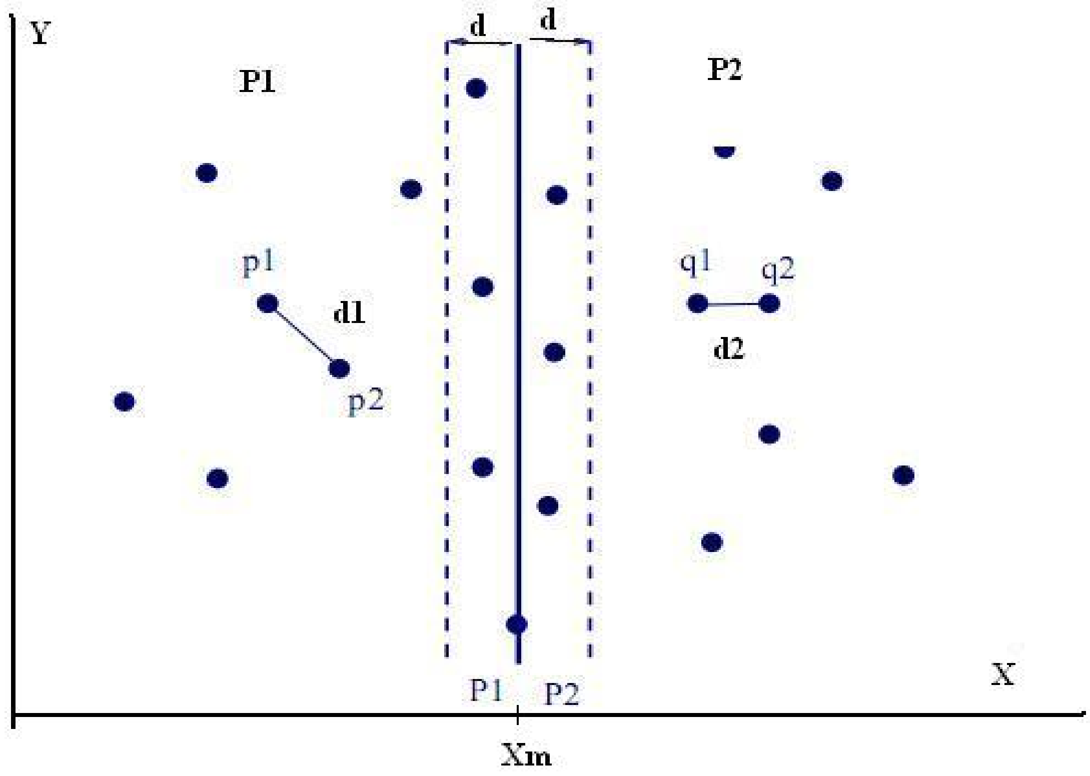
T(*n*) = 2T(*n*/2)+*n* ???

Ví dụ 3: Cặp điểm gần nhất

- **Phân tích giải pháp chia để trị**
  - Chia: Phân chia tập n điểm P thành 2 tập con P_1, P_2 (theo giá trị tăng dần của tọa độ x)
  - Trị: Tìm cặp điểm gần nhất trong mỗi tập P_1, P_2 với khoảng cách d_1, d_2
  - Kết hợp: So sánh và lấy ra cặp điểm gần hơn d = min(d_1,d_2)
  - Chú ý:
    - Có thể có cặp (1 điểm thuộc P_1, 1 điểm thuộc P_2) có khoảng cách ngắn hơn
    - Cần xem xét thêm các điểm nằm trong miền có khoảng cách

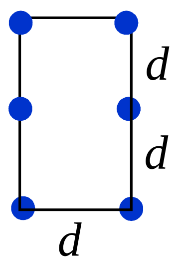
      ($X_m - d$, $X_m + d$) quanh điểm phân chia.

Ví dụ 3: Cặp điểm gần nhất



Ví dụ 3: Cặp điểm gần nhất

- **Độ phức tạp thuật toán**
  - Sắp xếp tập điểm theo tọa độ x: O($n$log$n$)
  - Chia: $P_1 \leq X_m \leq P_2$: O(n)
  - Trị: Đệ qui trên mỗi phần tìm d_1, d_2: 2T(n/2)
  - Kết hợp:
    - So sánh, tìm cặp điểm gần hơn *d:* O(*1*)
    - Tìm khoảng cách ngắn nhất giữa điểm $p_1 \in P_1$ và $p_2 \in P_1$: O(n)



O(n) vì: Với mỗi điểm nằm trong miền (X_m - d, X_m + d) được so sánh với 6 điểm xung quanh hình chữ nhật kích thước d x 2d

=> $T(n) = \max(O(n\log n), 2T(n/2) + O(n)) = O(n\log n)$