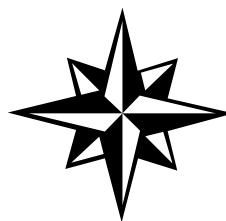


ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

**NGUYÊN LÝ & PHƯƠNG PHÁP
NGÔN NGỮ LẬP TRÌNH**



Tổng quan

- Mục đích của tối ưu hóa chương trình là nhằm giảm thiểu :
 - Thời gian
 - Thời gian chạy chương trình
 - Thời gian trả lời
 - Không gian
 - Không gian lưu trữ thứ cấp
 - Bộ nhớ chính
- Các chiến lược tối ưu hóa
 - Tối ưu theo mục tiêu được ưu tiên
 - Cân đối các mục tiêu:
 - Không gian và thời gian
 - Độ phức tạp

Tổng quan (tt)

- Khó khăn của vấn đề tối ưu hóa chương trình :
 - Có nhiều chiến lược khác nhau
- Phải quyết định :
 - Tối ưu hóa cái gì – không gian hay thời gian?
 - Tối ưu hóa ở đâu trong chương trình?

Khi nào cần tối ưu hóa?

- Khi chương trình không đạt được hiệu quả hợp lý về không gian và thời gian
- Khi có khả năng giảm thiểu độ phức tạp của chương trình

Một số kỹ thuật tối ưu hóa chương trình

Ví dụ minh họa

- **Bài toán:** hành trình của người đi bán hàng
 - Input: n điểm trên bản đồ
 - Output : một đường đi ngắn nhất, chỉ đi qua mỗi điểm một lần
- Vài giải pháp đơn giản:
 - Phát sinh tất cả các con đường có thể, sau đó xác định con đường ngắn nhất trong số đó.
 - Áp dụng phương pháp tìm kiếm “tham lam”.
 - ...

MỘT SỐ KỸ THUẬT TỐI ƯU HOÁ CHƯƠNG TRÌNH

TỐI ƯU HÓA CHƯƠNG TRÌNH

TỐI ƯU
THỜI GIAN

TỐI ƯU
KHÔNG GIAN

TỐI ƯU THỜI GIAN
VÀ KHÔNG GIAN

Tăng
không gian

Tăng
thời gian

Thuật toán **thay đổi**

Thuật toán **không đổi**

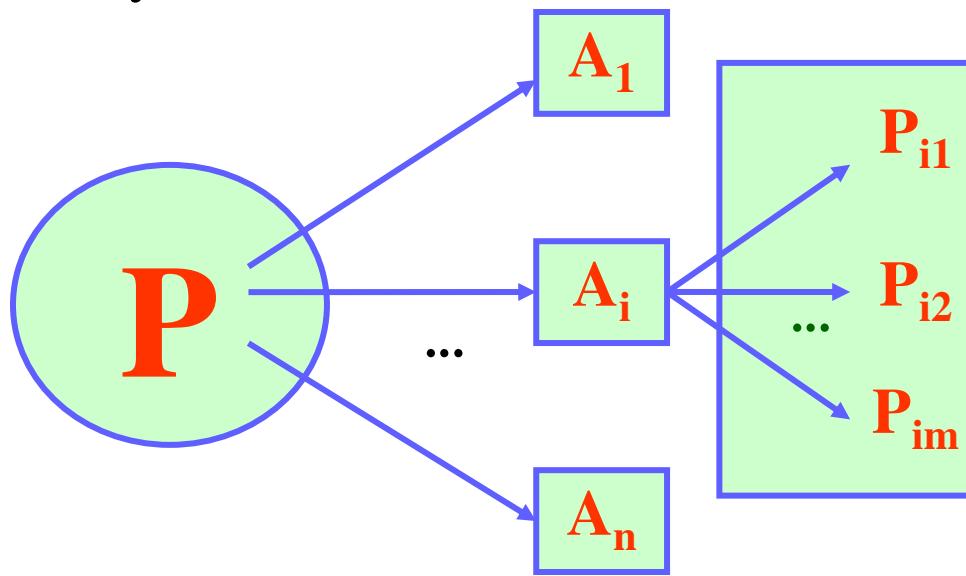
Thay đổi cấu trúc dữ liệu và
cấu trúc chương trình

Lặp,
rẽ nhánh,
Cấu trúc điều khiển

.....

TỐI ƯU HOÁ CHƯƠNG TRÌNH

- Thuật toán thay đổi



**CÁC KỸ THUẬT
TỐI ƯU THỜI GIAN THỰC HIỆN
CHƯƠNG TRÌNH**

TỐI ƯU HOÁ THỜI GIAN

1. Kỹ thuật tối ưu việc rẽ nhánh
2. Kỹ thuật tối ưu các vòng lặp
3. Tránh gọi lặp một thủ tục
4. Thay đổi, bố trí cấu trúc dữ liệu
5. Một số nguyên lý trong tối ưu hoá chương trình

TỐI ƯU VIỆC RẼ NHÁNH

- Không thể để các điều kiện A_i theo thứ tự ngẫu nhiên
- Phải sắp các A_i theo xác suất sai của A_i giảm dần
 - A_1 and A_2 and A_n
→ xác suất sai A_i giảm dần
 - A_1 or A_2 or A_n
→ xác suất đúng A_i giảm dần

TỐI ƯU CÁC VÒNG LẶP

1. Tách các lệnh không phụ thuộc vào chỉ số lặp ra khỏi vòng lặp.
- Lưu ý thời gian xử lý của các phép toán, để tối ưu thời gian thực hiện chương trình. $T_{\text{phép cộng}} < T_{\text{phép nhân}}$.
- Ví dụ 1:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
For i := 1 to 100 do Begin j := 2*k + i; A(j) := k-1; End;	n:= k+k ; m := k-1; For i := 1 to 100 do Begin j := n + i; A(j) := m; End;

TỐI ƯU CÁC VÒNG LẶP

- Ví dụ 2:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
<p>For i := 1 to ...</p> <p>Begin</p> <p>a:=b(i)+sin(x)</p> <p>b:=c(j)+cos(x);</p> <p>End;</p>	<p>t := sin(x);</p> <p>c := cos(x);</p> <p>For i := 1 to</p> <p>Begin</p> <p>a:=b(i)+t;</p> <p>b:=c(j)+c;</p> <p>End;</p>

TỐI ƯU CÁC VÒNG LẶP

2. Giảm số toán tử phức tạp trong vòng lặp nhờ các biến phụ.

Ví dụ:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
... vòng lặp ... $\sin(x) + \dots$...	$sx := \sin(x)$... vòng lặp ... $sx + \dots$...

TỐI ƯU CÁC VÒNG LẶP

3. Giảm số vòng lặp trong chương trình. Thực hiện nhiều công việc hơn trong mỗi vòng lặp.

Ví dụ 1:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
For i:=1 to 1000 do A(i) := 0;	For i:=1 to 500 step 2 do Begin A(i) := 0; A(i+1) := 0; End;

TỐI ƯU CÁC VÒNG LẶP

4. Vòng lặp nào có số lần lặp ít sẽ nằm ngoài vòng lặp có số lần lặp nhiều hơn.

For i := 1 to N do

 For j :=1 to M do

 For k :=1 to L do

ĐIỀU KIỆN: $N \leq M \leq L$

Chi phí khởi động vòng lặp ngoài > chi phí khởi động vòng lặp trong (2 lần)

TỐI ƯU CÁC VÒNG LẶP

NGUYÊN TẮC:

Ap dụng cho việc hoán đổi thứ tự các vòng lặp mà không làm sai ý nghĩa chương trình.

$$C_{NM} = N(C_1 + M.C_2) \rightarrow N.C_1 + MN.C_2$$

$$C_{MN} = M(C_1 + N.C_2) \rightarrow M.C_1 + MN.C_2$$

$$C_{NM} < C_{MN} \Leftrightarrow N < M$$

TỐI ƯU CÁC VÒNG LẶP

5. Thực hiện hợp nhất các vòng lặp có thể

Ví dụ:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
For i := 1 to 50 do For j := 1 to 50 do A(i,j) := 0; For k := 1 to 50 do A(k,k) := 1;	For i := 1 to 50 do Begin A(i,i) := 1; For j:= 1 to 50 do Begin A(i,j) := 0; End; End;

Giảm các kiểm tra điều kiện

- Ý tưởng: Nên có càng ít các kiểm tra điều kiện vòng lặp càng tốt, tốt nhất chỉ nên có một kiểm tra điều kiện vòng lặp.

Các phần tử cầm canh

- Ý tưởng: Nếu vòng lặp là một mảng tìm kiếm một chiều thì đặt một phần tử vào cuối dây cần tìm kiếm và loại bỏ điều kiện so sánh kiểm tra trong chỉ số vòng lặp
- Ví dụ: Tìm trong một phần tử trong đoạn (1..n)

i = 1;

while (item(i) != searchValue && i < n) {

i = i + 1;

}

if (item(i) == searchValue) { ... }

Các phần tử cầm canh

trở thành

i = 1;

save = item(n+1);

item(n+1) = searchValue;

while (item(i) != searchValue) {

 i = i+1;

}

item(n+1) = save;

if (i <= n) { ... }

Loại bỏ vòng lặp

- Ý tưởng: Loại bỏ (một phần) chi phí tính toán các chỉ số vòng lặp bằng cách xây dựng một biểu thức với các thành phần xác định
- Ví dụ:

sum = 0;

for (i = 1; i <= 5, i++) {

 sum = sum + x(i);

}

trở thành

sum = x(1) + x (2) + x(3) + x(4) + x(5);

Loại bỏ vòng lặp

- Khi chặn trên không biết, có một cách tiếp cận là:
 - (1) Giải phóng k bước lặp đầu tiên, tách k bản sao của phần thân với kiểm tra điều kiện.
 - (2) Đơn giản hóa kết quả bằng cách áp dụng các biến đổi phụ.

TRÁNH GỌI LẶP MỘT THỦ TỤC

Giảm tối đa việc gọi các thủ tục, hàm.

NGUYÊN LÝ: hãy làm **nhiều nhất** trong một vòng lặp, trong một thủ tục.

Ví dụ:

ĐOẠN CT GỐC	ĐOẠN CT TỐI ƯU
<p>Procedure P(i)</p> <p>.....</p> <p>end;</p> <p>For i := 1 to N do</p> <p>Call P(i);</p>	<p>Procedure P(i)</p> <p>For i := 1 to N do</p> <p>.....</p> <p>end;</p> <p>Call P;</p>

THAY ĐỔI, BỐ TRÍ DỮ LIỆU

1. Dùng biến phụ thay cho các biểu thức phải tính toán nhiều.

$$f = \frac{u + v + w + h}{k + l}$$

$$A = \frac{u + v + w + \dots}{k + l}$$

Đặt:

- $uvw = u + v + w$
- $kl = k + l$

Suy ra

$$f = \frac{uvw + h}{kl}$$

$$A = \frac{uvw + \dots}{kl}$$

THAY ĐỔI, BỐ TRÍ DỮ LIỆU

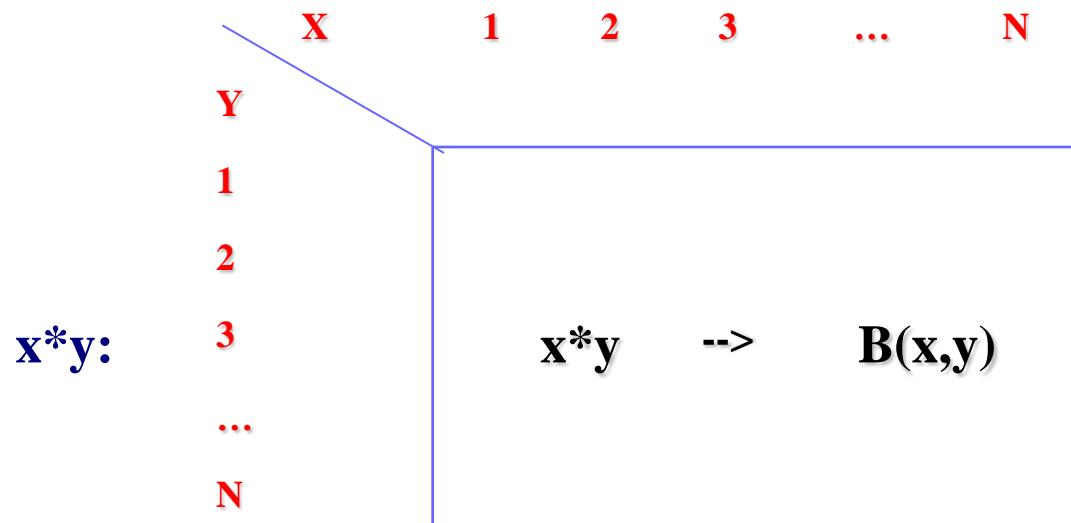
2. Dùng bảng truy cập (table-lookup) để tính toán (cách này áp dụng rất nhiều)

Dùng bảng truy cập thông minh → cố gắng lập bảng tính sẵn cho các trường hợp phổ biến.

$\text{case}(i) \in \{\text{phổ biến}\}$

→ table (i)

→ Tính case(i)



Các biến đổi logic

- Sử dụng các biểu thức tương đương
- Ngưng kiểm tra điều kiện khi đã biết kết quả
- Thứ tự kiểm tra các điều kiện
- Tính toán trước các hàm

Sử dụng các biểu thức tương đương

- Ý tưởng: Nếu việc đánh giá một biểu thức logic quá phức tạp, thay thế nó bằng một biểu thức tương đương nhưng đơn giản hơn
- Ví dụ 1:

```
if (square(x) > 0) {  
    [...]  
} else {  
    [...]  
}
```

Sử dụng các biểu thức tương đương *trở thành*

```
if (x !=0) { // square(x)>0 if x !=0
    [...]
} else {
    [...]
}
```

Sử dụng các biểu thức tương đương

- Ví dụ 2: (Giả sử có các số thực, $x, y > 0$)

if (**sqrt(x) < sqrt(y)**) {...}

trở thành

if (**x < y**) {...} // **sqrt(x) < sqrt(y)** if $x < y$

Ngưng kiểm tra nếu kín nếu bít trước kêt qua

- Ý tưởng: Không cần kiểm tra thêm nếu kín nếu không cần thiết
- Ví dụ 1:

```
if ( $p(x)$  &&  $b(x)$ ) { ... }
```

trở thành

```
if ( $p(x)$ ) {  
    if ( $b(x)$ ) {  
        }  
    }  
}
```

Ngưng kiểm tra điều kiện nếu biết được kết quả

Ví dụ 2:

```
if (p(x) || b(x)) { ... }
```

trở thành

```
if (p(x)) { ... } // ngưng nếu p(x)=true  
else if (b(x)) { ... }
```

Ngưng kiểm tra điều kiện nếu biết được kết quả

- Ví dụ 3:

```
sum = 0;
```

```
for (j = i; j < n; j++) {
```

```
    sum = sum + x(j);
```

```
}
```

```
if (sum > cutoff) { ... }
```

Ngưng kiểm tra điều kiện nếu biết được kết quả

Trở thành

```
sum = 0;  
j = i;  
while (j <= n && sum <= cutoff) {  
    sum = sum + x( j);  
    j = j + 1;  
}  
if (sum > cutoff) { ... }
```

Thứ tự kiểm tra các điều kiện

- Ý tưởng: Các kiểm tra logic có thể được sắp xếp sao cho các kiểm tra có **chi phí thấp và thường xuyên đúng** nằm ở trước các kiểm tra có chi phí cao và ít khi đúng
- Ví dụ: Giả sử các có các vị từ **a** và **b**, chi phí kiểm tra **a** 100mSec, và **b** là 1mSec.

```
if (a && b) {  
    p(x); [cost=101mSec * ...]  
} else {  
    p(y); [cost=101mSec * ...]  
}
```

MỘT SỐ NGUYÊN LÝ

■ NGUYÊN LÝ VANWIK

- Là nguyên lý phân cấp bộ nhớ
- Dữ liệu thường được truy cập nhiều phải được truy cập nhanh nhất

■ NGUYÊN LÝ ĐỐI XỨNG

- Là nguyên lý đối ngẫu, hay nguyên lý tính phần bù
- Tính một phần rồi lấy đối xứng, không cần tính toàn bộ

CÁC KỸ THUẬT TỐI ƯU HOÁ KHÔNG GIAN CHƯƠNG TRÌNH

TỐI ƯU HÓA KHÔNG GIAN

1. Nguyên lý nén dữ liệu
 - a) Giảm khoảng trống
 - b) Mã lắp
 - c) Mã hoá dựa vào tần suất
 - d) Mã nền
 - e) Ánh xạ co dữ liệu
 - f) Nén ảnh
2. Nguyên lý phân cấp bộ nhớ
3. Nguyên lý dùng công thức thay bộ nhớ

TỐI ƯU HOÁ KHÔNG GIAN

TỐI ƯU
KHÔNG GIAN CT

TỐI ƯU
BỘ NHỚ

TỐI ƯU KHÔNG
GIAN LÀM VIỆC

NGUYÊN LÝ NÉN DỮ LIỆU

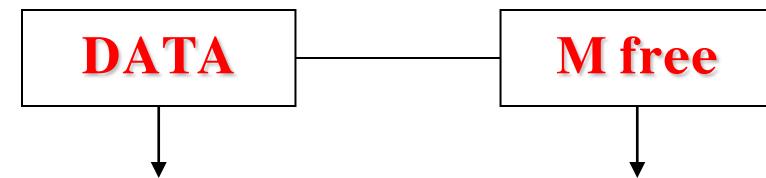
CHƯƠNG TRÌNH
 $P = \text{DATA} + \text{ALGORITHM}$

NÉN
CHƯƠNG TRÌNH

Bảo đảm dễ bảo quản
và bảo vệ

NÉN
DỮ LIỆU

Tiết kiệm bộ nhớ là
hàng đầu



Giảm thiểu vùng nhớ làm việc
cần cho chương trình

GIẢM KHOẢNG TRỐNG

Ma trận $M \times N$

1	0	0	1	0	1
0	1	1	..		
..	..				
..	..				

Biểu diễn
không còn
giá trị 0

Cách 1: biểu diễn danh sách
tọa độ các giá trị 1

$((i_1, j_1), (i_1, j_1), \dots, (i_1, j_1))$

Cách 2:

$(i_1, j_k, \dots) (\dots)$

$(i_1, j_1) \xrightarrow{\text{Tính}}$

Kiểm tra có trong danh sách

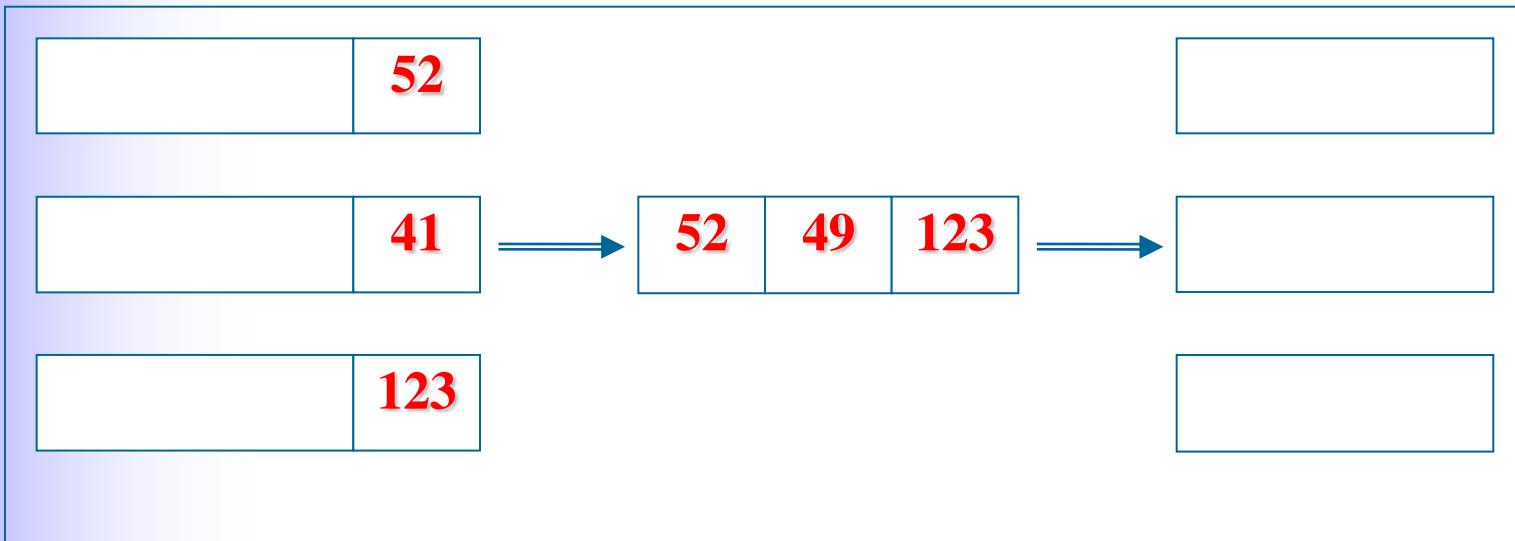
Test $(i, j) \in \text{List}(\text{data}) ?$

Trước khi tính cần kiểm tra có trong dữ liệu nén hay không? →
cần khoảng thời gian chấp nhận được.

Anh là bài toán đặc biệt

GIẢM KHOÁNG TRỐNG

Ví dụ: bài toán số ngắn



MÃ LẶP

- Có những dữ liệu thường xuyên lặp lại → dùng phép mã lặp Runlength-Coding để nén.
- Ví dụ:

5, 5, 5, 6, 7, 7, 7, → 3(5), 6, 4(7)

AAABBCCCD → A(3), B(2), C(3), D(1)
- Cơ sở dữ liệu very large
- Mã hóa khối dữ liệu: toàn khối dữ liệu mã hóa lại theo số lần xuất hiện của từ trong văn bản. Nếu từ xuất hiện nhiều nhất thì cho giá trị bé nhất → nén lại dữ liệu 0 1 đã có từ văn bản

Ví dụ: Cong hoa xa hoi chu nghia Viet Nam

Cong xuất hiện nhiều nhất 0

.....

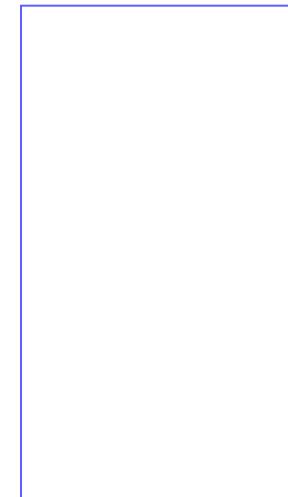
hoa xuat hien 1 lan 1
→ 000101

MÃ HÓA DỰA VÀO TẦN SUẤT

BẢN TEXT



BẢN ENCRYPTION



e	$\$_1$	15%
i	$\$_2$	10%
....	$\$_n$	0.01%

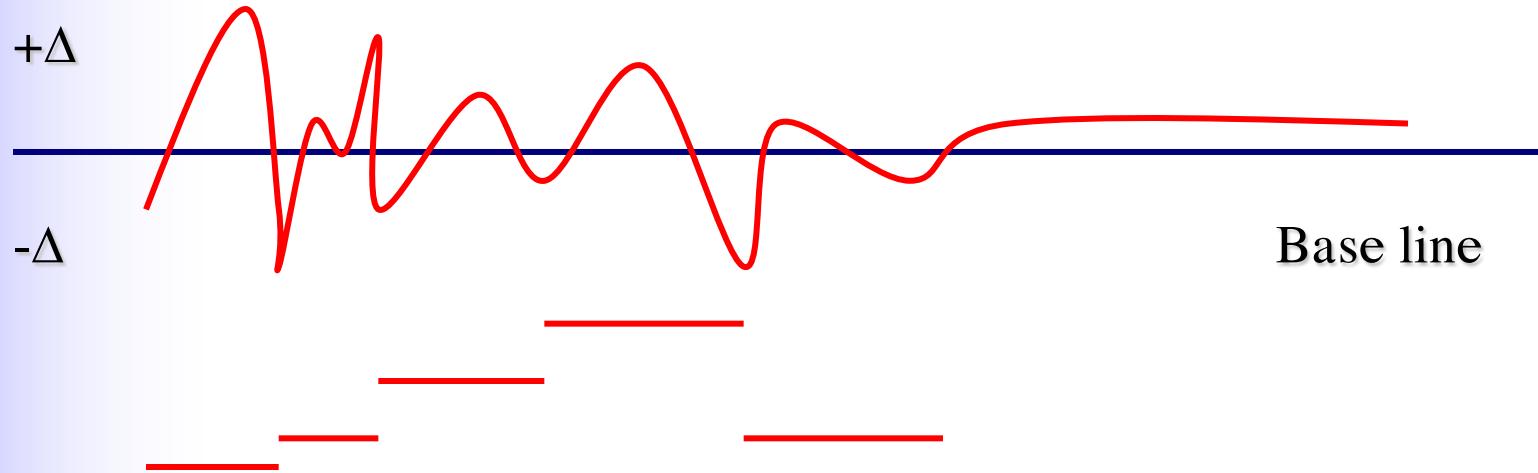
MÃ NỀN

Ví dụ:

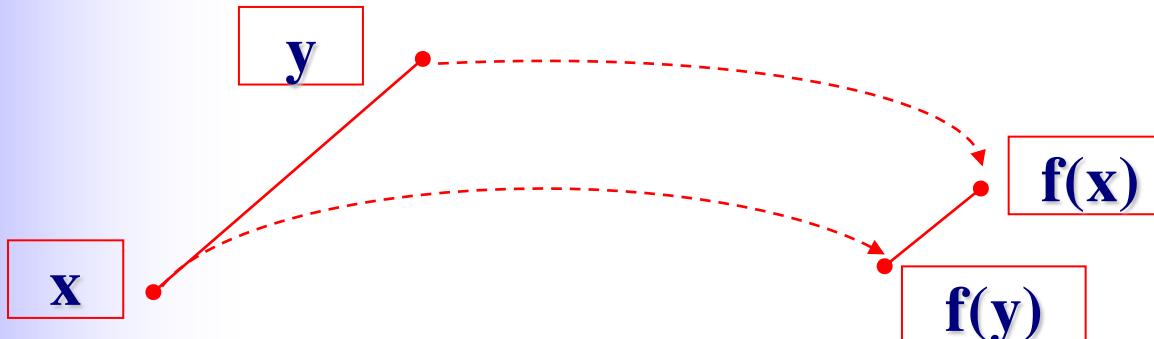
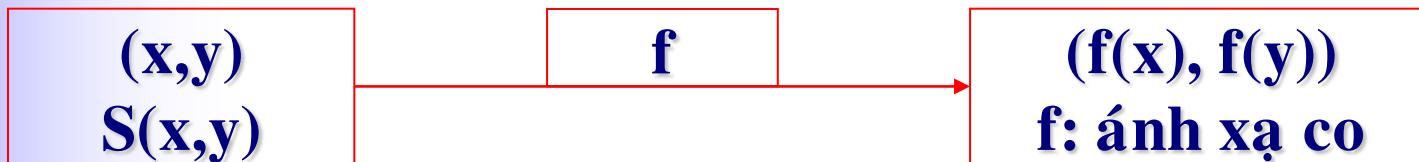
157, 158, 162, 174,

(+100) 57, 58, 62, 74.....

- Xử lý tín hiệu tiếng nói → xem xét dữ liệu để chọn nền cho dữ liệu



ÁNH XẠ CO DỮ LIỆU



$$S(x,y) \geq \varepsilon \cdot f$$

$$\|(x, y)\| \geq \varepsilon \cdot \|f(x), f(y)\|$$

$fk(x,y) \rightarrow fk(fk(x), fk(y)) \rightarrow \dots$

ÁNH XẠ CO DỮ LIỆU

Ví dụ:

$$f = 5x + 4y + 6$$

1000 lần

$$x^*, y^*$$

- Một ánh xạ co không phụ thuộc vào input

$$\begin{array}{c} \forall(x,y) \\ \downarrow \\ f \\ \downarrow \\ x^*, y^* \end{array}$$

ÁNH XẠ CO DỮ LIỆU

Thay vì lưu trữ ảnh màu có 256 màu có kích thước rất lớn thì chỉ lưu trữ ánh xạ co.

Lớp ánh xạ co (affine)

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

(a, b, c, d, e, f)

$$(x, y) \rightarrow (x', y')$$

$$(x'_1, y'_1) \rightarrow (x_1, y_1)$$

$$(x'_2, y'_2) \rightarrow (x_2, y_2)$$

$$(x'_3, y'_3) \rightarrow (x_3, y_3)$$

*1

*2

*3

*1

*3

*2

Nhỏ

Lớn

ÁNH XẠ CO DỮ LIỆU

Để tìm được 6 hệ số cần phải :

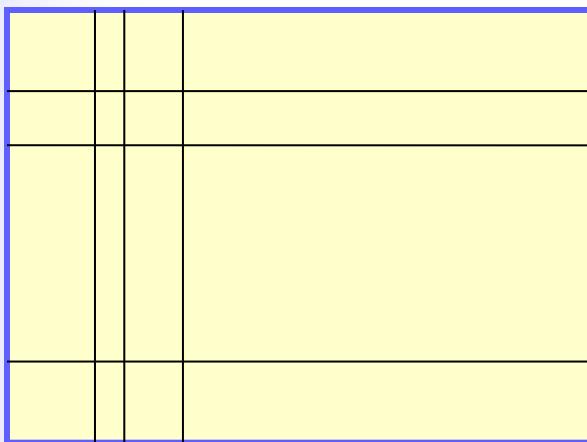
- Tìm được 3 cặp điểm giữa 2 bên
- Tìm được ánh xạ co
- Tìm được 6 phương trình

$$\left\{ \begin{array}{l} x'_1 = ax_1 + by_1 + c \\ y'_1 = dx_1 + ey_1 + f \\ .. \\ .. \\ x'_3 = ax_3 + by_3 + c \\ y'_3 = dx_3 + ey_3 + f \end{array} \right.$$

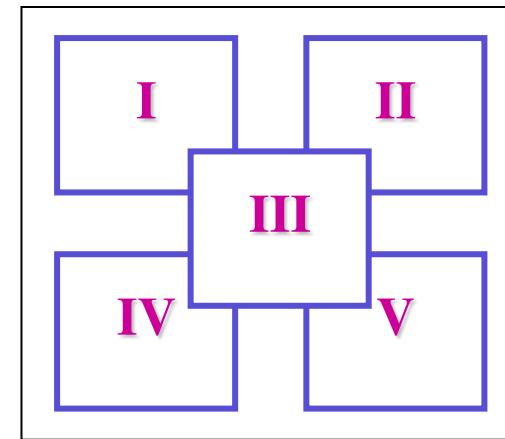
→ a, b, c, d, f

NÉN ẢNH

Từ ảnh gốc chia theo lưới vuông, tam giác, chữ nhật

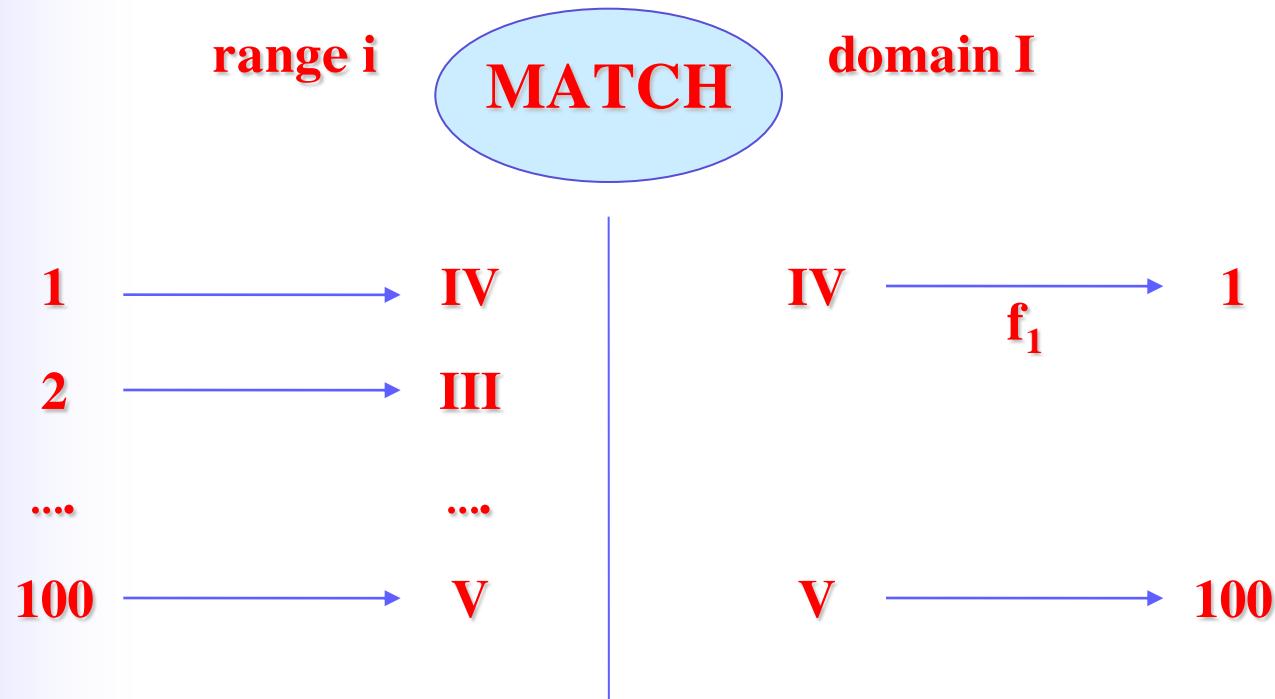


Chia thành các
range không giao
nhau



Ảnh gốc được chia
thành các domain,
các domain có thể
giao nhau

NÉN ẢNH



NÉN ẢNH

NGUYÊN LÝ TỰ ĐỒNG DẠNG: mọi đối tượng đều tự đồng dạng với nhau

Ảnh nén rất nhỏ, nên cần thời gian tìm sự đồng dạng lớn.

Có 3 dạng dữ liệu cần quan tâm để nén: văn bản, số liệu (ma trận,...), hình ảnh, tiếng nói

NGUYÊN LÝ PHÂN CẤP BỘ NHỚ

NGUYÊN LÝ VANWIJK: dữ liệu thường được truy cập phải được truy cập tốt nhất

TỪ ĐIỂN



TỪ ĐIỂN THÔNG DỤNG



90% - 3000 từ

997000 từ



BN trong

BN ngoài

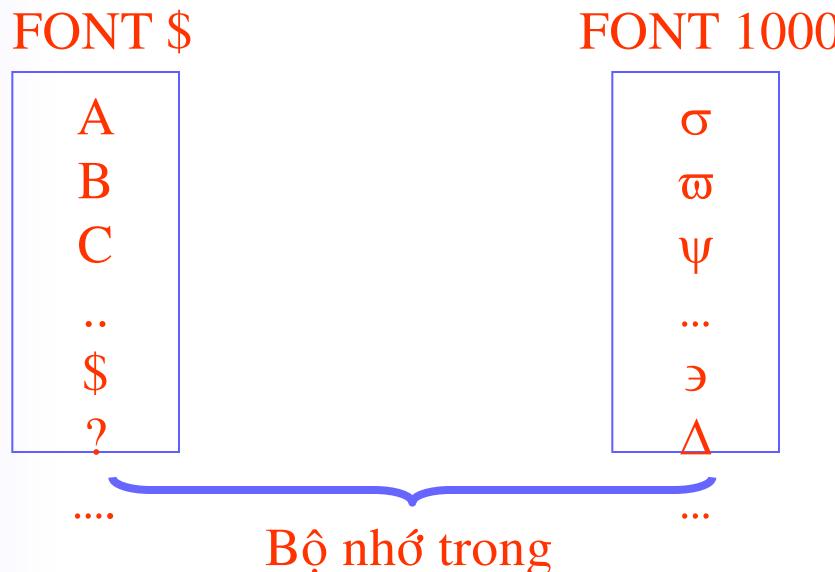
NGUYÊN LÝ PHÂN CẤP BỘ NHỚ

- Bài toán với khối lượng dữ liệu lớn, đa dạng**

Ví dụ: sắp xếp 1 000 000 số

2000 → thứ tự cục bộ → thứ tự toàn phần

Ví dụ: bài toán nhận dạng → phân cấp theo mẫu tự hay xuất hiện
phân cấp theo font thường dùng



NGUYÊN LÝ DÙNG CÔNG THỨC THAY BỘ NHỚ

- Dùng bộ nhớ thay công thức tính

Ví dụ:

Dãy Fibonaci: 1, 1, 2, 3, 5, 8, 13

$$U_n = U_{n-1} + U_{n-2}$$

→ U_{1000} bùng nổ không gian vùng nhớ

- Mọi đệ quy đều có thể đưa về công thức đúng, gần đúng 1 lần

Ví dụ: trắc địa

1/ x, y, z,..... θ, σ, \dots

2/

.....

A, B



A, t (A → B)