



Chuong 1 - tài liệu thuật toán nâng cao Chương 1

Cấu trúc dữ liệu và giải thuật (Học viện Công nghệ Bưu chính Viễn thông)



Scan to open on Studeersnel



CHUYÊN ĐỀ THUẬT TOÁN NÂNG CAO

NỘI DUNG:

CHƯƠNG 1. MỞ ĐẦU

- 1.1. Khái niệm thuật toán
- 1.2. Biểu diễn thuật toán
- 1.3. Độ phức tạp thuật toán
- 1.4. Độ phức tạp chương trình
- 1.5. Một số thuật toán cơ bản
- 1.6. CASE STUDY

CHƯƠNG 2. MÔ HÌNH DUYỆT VÀ ĐỆ QUI

- 2.1. Thuật toán duyệt
- 2.2. Thuật toán sinh
- 2.3. Thuật toán đệ qui
- 2.4. Thuật toán quay lui
- 2.5. Khử đệ qui
- 2.6. CASE STUDY

CHƯƠNG 3. Mô hình thuật toán tham lam

3.1. Giới thiệu giải thuật

3.2. Bài toán Activities Selection

3.3. Bài toán N-Ropes

3.4. Bài toán sắp đặt lại xâu ký tự

3.5. Bài toán tìm cây khung nhỏ nhất

3.6. CASE STUDY

CHƯƠNG 4. Thuật toán chia và trị

4.1. Giới thiệu giải thuật

4.2. Thuật toán cộng

4.3. Thuật toán nhân

4.4. Giải thuật Binary-Search

4.5. Giải thuật Quick Sort

4.6. Giải thuật Merge Sort

4.7. CASE STUDY

CHƯƠNG 5. Mô hình thuật toán qui hoạch động

NỘI DUNG:

5.1. Giới thiệu thuật toán

5.2. Bài toán chia số

5.3. Bài toán cái túi

5.4. Bài toán Activities Selection

5.5. Một số bài toán qui hoạch động điển hình

5.6. CASE STUDY

CHƯƠNG 6. Mô hình nhánh cận

6.1. Mô tả thuật toán

6.2. Bài toán cái túi

6.3. Bài toán người du lịch

6.4. Bài toán lập lịch

6.5. Một số bài toán điển hình

6.6. CASE STUDY

CHƯƠNG 7. Một số thuật toán tìm kiếm mẫu

7.1. Mô tả thuật toán

7.2. Tìm kiếm mẫu từ trái qua phải

7.3. Tìm kiếm mẫu từ phải qua trái

7.4. Tìm kiếm mẫu từ vị trí xác định

7.5. Tìm kiếm mẫu từ vị trí bất kỳ

7.6. CASE STUDY

TÀI LIỆU THAM KHẢO:

- [1] Robert Sedgewick, “Algorihms”, McGraw Hill Company, 2006, Vol 1, 2, 3, 4.
- [2] N. Knuth, “The Art of Programming”, “Algorihms”, McGraw Hill Company, 2006, Vol 1, 2, 3, 4.
- [3] Robert Sedgewick, “Cẩm nang thuật toán”, Nhà xuất bản khoa học kỹ thuật Hà Nội, 2004, Vol 1, 2, 3, 4.
- [4] Lê Minh Hoàng, “Bài giảng các chuyên đề”, Nhà xuất bản khoa học kỹ thuật Hà Nội, 2008.

YÊU CẦU:

1. Hiểu phương pháp biểu diễn thuật toán.
2. Đánh giá độ phức tạp thuật toán.
3. Cài đặt thuật toán bằng ngôn ngữ C++, JAVA.
4. Hoàn thành CASE STUDY theo mỗi chương.

PHƯƠNG PHÁP ĐÁNH GIÁ:

- CASE STUDY 1 : 10%
- CASE STUDY 2 : 10%
- CASE STUDY 3 : 10%
- CASE STUDY 4 : 10%
- CASE STUDY 5 : 10%
- CASE STUDY 6 : 10%
- Chuyên cần : 20%
- Trả lời vấn đáp : 20%



CHUYÊN ĐỀ THUẬT TOÁN NÂNG CAO

NỘI DUNG:

CHƯƠNG 1. MỞ ĐẦU

1.1. Khái niệm thuật toán

1.2. Biểu diễn thuật toán

1.3. Độ phức tạp thuật toán

1.4. Một số thuật toán cơ bản

1.5. CASE STUDY

1.1. Thuật toán và giải thuật

1.1.1. Định nghĩa thuật toán (Algorithm): Thuật toán F giải bài toán P là dãy các thao tác sơ cấp F_1, F_2, \dots, F_N trên tập dữ kiện đầu vào (Input) để đưa ra được kết quả ra (Output).

$$F_1 F_2 \dots F_N (\text{Input}) \rightarrow \text{Output}.$$

- $F = F_1 F_2 \dots F_N$ được gọi là thuật toán giải bài toán P. Trong đó, mỗi F_i chỉ là các phép tính toán số học hoặc logic.
- Input được gọi là tập dữ kiện đầu vào (dữ liệu đầu vào).
- Output là kết quả nhận được sau khi thực hiện thuật toán F trên tập Input.

Ví dụ. Thuật toán tìm USCLN(a, b).

```
Int      USCLN ( int a, int b ) {  
    while (b!=0) {  
        x = a % b; a = b; b =x;  
    }  
    return(a);  
}
```

1.1.2. Các đặc trưng của thuật toán:

Một thuật toán cần thỏa mãn các tính chất dưới đây:

- **Tính đơn định.** Ở mỗi bước của thuật toán, các thao tác sơ cấp phải hết sức rõ ràng, không gây nên sự lộn xộn, nhập nhằng, đa nghĩa. Thực hiện đúng các bước của thuật toán trên tập dữ liệu vào, chỉ cho duy nhất một kết quả ra.
- **Tính dừng.** Thuật toán không được rơi vào quá trình vô hạn. Phải dừng lại và cho kết quả sau một số hữu hạn các bước.
- **Tính đúng.** Sau khi thực hiện tất cả các bước của thuật toán theo đúng qui trình đã định, ta phải nhận được kết quả mong muốn với mọi bộ dữ liệu đầu vào. Kết quả đó được kiểm chứng bằng yêu cầu của bài toán.
- **Tính phổ dụng.** Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong lớp các bài toán cùng loại và có thể làm việc trên nhiều loại dữ liệu khác nhau.
- **Tính khả thi.** Thuật toán phải dễ hiểu, dễ cài đặt, thực hiện được trên máy tính với thời gian cho phép.

1.1.3. Khái niệm giải thuật

Giải thuật là khái niệm mở rộng của thuật toán, trong đó tính “*xác định*” được làm mềm đi. Ví dụ dưới đây sẽ minh họa cho điều này.

Ví dụ. Tìm USCLN (a, b).

```
Int      USCLN ( int a, int b ) {  
    if ( a > b ) USCLN( a-b, b );  
    else if ( a < b ) USCLN( a, b-a );  
    return(a);  
}
```

Giả sử ta tìm USCLN(8, 12)
= USCLN(8, 4); //Chưa xác định
= USCLN (4,4); // Chưa xác định
= 4.

Chính vì lý do trên khái niệm giải thuật được thay thế cho khái niệm thuật toán. Tuy vậy, do thói quen người ta vẫn sử dụng đồng nhất các khái niệm này. Từ nay về sau, ta cũng sử dụng chung khái niệm thuật toán và giải thuật là giống nhau và đều có nghĩa: Algorithm, Method, Procedure, Process.

1.2. Phương pháp biểu diễn thuật toán:

Thông thường, để biểu diễn một thuật toán ta có thể sử dụng các phương pháp sau:

- **Biểu diễn bằng ngôn ngữ tự nhiên.** Ngôn ngữ tự nhiên là phương tiện giao tiếp giữa con người với con người. Ta có thể sử dụng chính ngôn ngữ này vào việc biểu diễn thuật toán.
- **Ngôn ngữ hình thức.** Ngôn ngữ hình thức là phương tiện giao tiếp trung gian giữa con người và hệ thống máy tính. Ví dụ ngôn ngữ sơ đồ khối, ngôn ngữ tựa tự nhiên, ngôn ngữ đặc tả. Đặc điểm chung của các loại ngôn ngữ này là việc sử dụng nó rất gần với ngôn ngữ máy tính.
- **Ngôn ngữ máy tính.** Là phương tiện giao tiếp giữa máy tính và máy tính. Trong trường hợp này ta có thể sử dụng bất kỳ ngôn ngữ lập trình nào để mô tả thuật toán.

Ghi chú. Trong các phương pháp biểu diễn thuật toán, phương pháp biểu diễn bằng ngôn ngữ hình thức được sử dụng rộng rãi vì nó gần với ngôn ngữ tự nhiên và không phụ thuộc vào ngôn ngữ máy tính.

Ví dụ. Biểu diễn thuật toán tìm USCLN (a, b).

Biểu diễn bằng ngôn ngữ tự nhiên.

Đầu vào (Input). Hai số tự nhiên a, b.

Đầu ra (Output). Số nguyên u lớn nhất để a và b đều chia hết cho u.

Thuật toán (Euclidean Algorithm):

Bước 1. Đưa vào hai số tự nhiên a và b.

Bước 2. Nếu $b \neq 0$ thì chuyển đến bước 3, nếu $b=0$ thì thực hiện bước 4.

Bước 3. Đặt $r = a \bmod b$; $a = b$; $b = r$; Quay quay trở lại bước 2.

Bước 4 (Output). Kết luận $u=a$ là số nguyên cần tìm.

Biểu diễn bằng ngôn ngữ hình thức.

Thuật toán Euclidean:

Đầu vào (Input): $a \in N$, $b \in N$.

Đầu ra (Output): $s = \max \{ u \in N : a \bmod u = 0 \text{ and } b \bmod u = 0 \}$.

Format : $s = Euclidean(a, b)$.

Actions :

```
while ( $b \neq 0$ ) do  
     $r = a \bmod b$ ;  $a = b$ ;  $b = r$ ;  
endwhile;  
return( $a$ );
```

Endactions.

Ví dụ. Biểu diễn thuật toán tìm USCLN (a, b).

Biểu diễn bằng ngôn ngữ máy tính (C++).

```
Int      USCLN( int a, int b) {  
    while ( b != 0 ) {  
        r = a % b; a = b; b = r;  
    }  
    return(a);  
}
```

1.3. Độ phức tạp tính toán

Một bài toán có thể thực hiện bằng nhiều thuật toán khác nhau. Chọn giải thuật nhanh nhất giải bài toán là một nhu cầu của thực tế. Vì vậy ta cần phải có sự ước lượng cụ thể để minh chứng bằng toán học mức độ nhanh chậm của mỗi giải thuật.

1.3.1. Khái niệm độ phức tạp thuật toán

Thời gian thực hiện một giải thuật bằng chương trình máy tính phụ thuộc vào các yếu tố:

- Kích thước dữ liệu vào: Dữ liệu càng lớn thì thời gian xử lý càng chậm.
- Phần cứng máy tính: máy có tốc độ cao thực hiện nhanh hơn trên máy có tốc độ thấp. Tuy vậy, yếu tố này không ảnh hưởng đến quá trình xác định thời gian thực hiện của thuật toán nếu xem xét thời gian thực hiện thuật toán như một hàm của độ dài dữ liệu $T(n)$.

Tổng quát, cho hai hàm $f(x), g(x)$ xác định trên tập các số nguyên dương hoặc tập các số thực vào tập các số thực. Hàm $f(x)$ được gọi là $O(g(x))$ nếu tồn tại một hằng số $C > 0$ và n_0 sao cho:

$$|f(x)| \leq C \cdot |g(x)| \text{ với mọi } x \geq n_0.$$

Điều này có nghĩa với các giá trị $x \geq n_0$ hàm $f(x)$ bị chặn trên bởi hằng số C nhân với $g(x)$. Nếu $f(x)$ là thời gian thực hiện của một thuật toán thì ta nói giải thuật đó có cấp $g(x)$ hay độ phức tạp thuật toán là $O(g(x))$.

Ghi chú. Các hằng số C, n0 thỏa mãn điều kiện trên là không duy nhất. Nếu có đồng thời $f(x)$ là $O(g(x))$ và $h(x)$ thỏa mãn $g(x) < h(x)$ với $x > n0$ thì ta cũng có $f(x)$ là $O(h(n))$.

Ví dụ 1. Cho $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

Trong đó, ai là các số thực ($i = 0, 1, 2, \dots, n$). Khi đó $f(x) = O(x^n)$.

Chứng minh. Thực vậy, với mọi $x > 1$:

$$\begin{aligned}|f(x)| &= |a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0| \\&\dots \leq |a_n| x^n + |a_{n-1}| x^{n-1} + \dots + |a_1| x + |a_0| \\&\dots \leq |a_n| x^n + |a_{n-1}| x^n + \dots + |a_1| x^n + |a_0| x^n \\&\dots \leq x^n (|a_n| + |a_{n-1}| + \dots + |a_1| + |a_0|) \\&\dots \leq C \cdot x^n = O(x^n).\end{aligned}$$

$$C = (|a_n| + |a_{n-1}| + \dots + |a_1| + |a_0|)$$

Ví dụ 2. Tìm độ phức tạp thuật toán sắp xếp kiểu Bubble-Sort?

```
Void Bubble-Sort ( int A[], int n ) {  
    for ( i=1; i<n; i++ ) {  
        for ( j = i+1; j<=n; j++ ){  
            if (A[i] > A[j]) {  
                t = A[i]; A[i] = A[j]; A[j] = t;  
            }  
        }  
    }  
}
```

Lời giải. Sử dụng trực tiếp nguyên lý cộng ta có:

- Với $i = 1$ ta cần sử dụng $n-1$ phép so sánh $A[i]$ với $A[j]$;
- Với $i = 2$ ta cần sử dụng $n-1$ phép so sánh $A[i]$ với $A[j]$;
-
- Với $i = n-1$ ta cần sử dụng 1 phép so sánh $A[i]$ với $A[j]$;

Vì vậy tổng số các phép toán cần thực hiện là:

$$S = (n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2 \leq n^2 = O(n^2).$$

Ghi chú. Độ phức tạp thuật toán cũng là số lần thực hiện phép toán tích cực. Phép toán tích cực là phép toán thực hiện nhiều nhất đối với thuật toán.

1.3.2. Qui tắc xác định độ phức tạp thuật toán

Qui tắc tổng: Nếu $f_1(x)$ có độ phức tạp là $O(g_1(x))$ và $f_2(x)$ có độ phức tạp là $O(g_2(x))$ thì độ phức tạp của $(f_1(x) + f_2(x))$ là $O(\text{Max}(g_1(x), g_2(x)))$.

Chứng minh.

- Vì $f_1(x)$ có độ phức tạp là $O(g_1(x))$ nên tồn tại hằng số C_1 và k_1 sao cho $|f_1(x)| \leq |g_1(x)|$ với mọi $x \leq k_1$;
- Vì $f_2(x)$ có độ phức tạp là $O(g_2(x))$ nên tồn tại hằng số C_2 và k_2 sao cho $|f_2(x)| \leq |g_2(x)|$ với mọi $x \leq k_2$;
- Ta lại có :

$$\begin{aligned}|f_1(x) + f_2(x)| &\leq |f_1(x)| + |f_2(x)| \\&\leq C_1|g_1(x)| + C_2|g_2(x)| \\&\leq C|g(x)| \text{ với mọi } x > k;\end{aligned}$$

Trong đó, $C = C_1 + C_2$; $g(x) = \max(g_1(x), g_2(x))$; $k = \max(k_1, k_2)$.

Tổng quát. Nếu độ phức tạp của $f_1(x), f_2(x), \dots, f_m(x)$ lần lượt là $O(g_1(x)), O(g_2(x)), \dots, O(g_n(x))$ thì độ phức tạp của $f_1(x) + f_2(x) + \dots + f_m(x)$ là $O(\max(g_1(x), g_2(x), \dots, g_m(x)))$.

Qui tắc nhân: Nếu $f(x)$ có độ phức tạp là $O(g(x))$ thì độ phức tạp của $f^n(x)$ là $O(g^n(x))$.
Trong đó:

$$f^n(x) = f(x).f(x)....f(x). //n lần f(x).$$

$$g^n(x) = g(x).g(x)...g(x).//n lần g(x)$$

Nói cách khác, đoạn chương trình P có thời gian thực hiện $T(n)= O(f(n))$. Khi đó, nếu thực hiện $k(n)$ lần đoạn chương trình P với $k(n)$ là $O(g(n))$ thì độ phức tạp tính toán là $O(f(n). g(n))$.

Chứng minh. Thật vậy theo giả thiết $f(x)$ là $O(g(x))$ nên tồn tại hằng số C và k sao cho với mọi $x>k$ thì $|f(x)|\leq C.|g(x)|$. Ta có:

$$\begin{aligned}|f^n(x)| &= \left|f^1(x).f^2(x)...f^n(x)\right| \\&\leq \left|C.g^1(x).C.g^2(x)...C.g^n(x)\right| \\&\leq C^n \left|g^n(x)\right| = O(g^n(x))\end{aligned}$$

Một số tính chất của độ phức tạp thuật toán:

- Với $P(n)$ là một đa thức bậc k thì $O(P(n)) = O(n^k)$. Vì thế ta nói, một thuật toán có độ phức tạp cấp đa thức là $O(n^k)$.
- Với a, b là hai cơ số tùy ý và $f(n)$ là một hàm xác định dương thì $\log_a f(n) = \log_a b \cdot \log_b(f(n))$. Vì vậy độ phức tạp thuật toán cấp logarit được ký hiệu là $O(\log(f(n)))$ mà không cần quan tâm đến cơ số.
- Nếu độ phức tạp thuật toán là hằng số, nghĩa là thời gian tính toán không phụ thuộc vào độ dài dữ liệu được ký hiệu là $O(1)$.
- Một giải thuật có cấp 2^n , $n!$, n^n được gọi là giải thuật hàm mũ. Những giải thuật này thường có tốc độ rất chậm.
- Độ phức tạp tính toán của một đoạn chương trình P chính bằng số lần thực hiện một phép toán tích cực. Trong đó, phép toán tích cực trong một đoạn chương trình là phép toán mà số lần thực hiện nó không ít hơn các phép toán khác.

CÁC DẠNG HÀM ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN

Dạng đánh giá	Tên gọi
$O(1)$	Hằng số
$O(\lg \lg n)$	Log log
$O(\lg n)$	Logarithm
$O(n)$	Tuyến tính
$O(n^2)$	Bậc hai
$O(n^3)$	Bậc 3
$O(n^m)$	Đa thức
$O(m^n)$	Hàm mũ
$O(n!)$	Giai thừa

1.4. Độ phức tạp của các cấu trúc lệnh

Để đánh giá độ phức tạp của một thuật toán đã được mã hóa thành chương trình máy tính ta thực hiện theo qui tắc sau.

Độ phức tạp hằng số O(1): đoạn chương trình không chứa vòng lặp hoặc lời gọi đệ qui có tham biến là một hằng số.

Ví dụ. Đoạn code dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=c; i++) {  
    <Tập các chỉ thị có độ phức tạp O(1)>;  
}
```

Độ phức tạp O(n): Độ phức tạp của hàm hoặc đoạn code là O(n) nếu biến trong vòng lặp tăng hoặc giảm bởi một hằng số c.

Ví dụ. Đoạn code dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=n; i = i + c ) {  
    <Tập các chỉ thị có độ phức tạp O(1)>;  
}  
for (i=n; i>0; i = i - c ){  
    <Tập các chỉ thị có độ phức tạp O(1)>;  
}
```

Độ phức tạp hằng số $O(n^c)$: Độ phức tạp của các vòng lặp lồng nhau bằng lũy thừa của số lần lồng nhau.

Ví dụ. Đoạn code dưới đây có độ phức tạp $O(n^2)$.

```
for (i=1; i<=n; i = i + c ) {  
    for (j=1; j<=n; j = j + c ){  
        <Tập các chỉ thị có độ phức tạp O(1)>;  
    }  
}  
for (i = n; i >0 ; i = i - c ) {  
    for (j = i- 1; j>1; j = j -c ){  
        <Tập các chỉ thị có độ phức tạp O(1)>;  
    }  
}
```

Độ phức tạp logarit $\text{Log}(n)$: Độ phức tạp của vòng lặp là $\text{log}(n)$ nếu biến lặp được chia hoặc nhân với một hằng số c.

Ví dụ. Đoạn code dưới đây có độ phức tạp $\text{Log}(n)$.

```
for (i=1; i <=n; i = i *c ){  
    <Tập các chỉ thị có độ phức tạp O(1)>;  
}  
for (j=n; j >0 ; j = j / c ){  
    <Tập các chỉ thị có độ phức tạp O(1)>;  
}
```

Độ phức tạp hằng số Log Log(n): nếu biến lặp được nhân hoặc chia cho một hàm mũ.

Ví dụ. Đoạn code dưới đây có độ phức tạp Log Log(n).

```
for (i=1; j<=n; j*= Pow(i, c )){  
    <Tập các chỉ thị có độ phức tạp O(1)>  
}  
for (j=n; j>=0; j = j- Function(j) ){  
    //Function(j) =sqrt(j) hoặc lớn hơn 2.  
    <Tập các chỉ thị có độ phức tạp O(1)>  
}
```

1.6. Một số thuật toán cơ bản

Thuật toán Euclid tìm ước số chung lớn nhất của hai số a, b.

```
Int    USCLN( a, b ) {  
    x = a; y =b;  
    while (y !=0 ) {  
        r = x mod y;  
        x = y;  
        y = r;  
    }  
    return(x);  
}
```

Thuật toán biểu diễn n ở hệ cơ số b.

```
Void Digit ( int n, int b ){  
    int coso[MAX], sodu, k=1;  
    while ( n!=0 ) {  
        sodu=n%10;  
        n = n/10;  
        coso[k++] =sodu;  
    }  
    for (int j=k-1; j>=1; j--) //đảo ngược mảng cơ số  
        cout<<coso[j]<<“ “;  
}
```

Thuật toán cộng hai số nguyên.

Input:

$a = (a_{n-1}, a_{n-2}, \dots, a_0)_2$ là khai triển nhị phân của a.

$b = (b_{n-1}, b_{n-2}, \dots, b_0)_2$ là khai triển nhị phân của b.

Output:

$s = (s_{n-1}, s_{n-2}, \dots, s_0)_2$ là khai triển nhị phân của $a+b$.

Format : $s = \text{Addition}(a,b);$

Actions:

$c = 0; //c số nhớ$

$\text{for } (j=0; j \leq n-1; j++)\{$

$d = (a_j + b_j + c)/2;$

$s_j = a_j + b_j + c - 2d;$

$c = d;$

$\}$

$s_n = c;$

$\text{return}(s_n, s_{n-1}, \dots, s_0);$

$\}$

1.5. Tính đúng đắn của thuật toán dựa vào dữ liệu

Khi hoàn thành việc xây dựng một thuật toán thì thuật toán đó cần phải được kiểm nghiệm. Phương pháp kiểm nghiệm thuật toán phổ biến hiện nay thường được sử dụng trong tin học là kiểm nghiệm dựa vào dữ liệu. Để có thể kiểm nghiệm thuật toán dựa vào dữ liệu ta cần tiến hành xây dựng các bộ dữ liệu Test. Các bộ dữ liệu Test cần thỏa mãn:

- Kiểm tra được kết quả thực hiện của thuật toán đối với dữ liệu nhỏ.
- Kiểm tra được kết quả thực hiện của thuật toán đối với dữ liệu lớn.
- Kiểm tra được kết quả của thuật toán đối với các trường hợp đặc biệt.
- Kiểm tra được độ phức tạp của thuật toán dựa vào thời gian thực hiện mỗi test.
- Kiểm tra được kỹ thuật cài đặt thuật toán của người lập trình.

Ví dụ 1. Sử dụng cấu trúc dữ liệu thích hợp, hãy viết chương trình chuyển đổi số tự nhiên n thành số ở hệ cơ số b ($2 \leq b \leq 32$).

Dữ liệu vào (Input) cho bởi file data.in theo khuôn dạng:

- Dòng đầu tiên ghi lại số tự nhiên K là số lượng các test ($k \leq 100$).
- K dòng kế tiếp ghi lại mỗi dòng một test. Mỗi test bao gồm một cặp số n, b . Hai số được viết cách nhau một vài khoảng trắng.

Kết quả ra (Output): ghi lại K dòng trong file ketqua.out, mỗi dòng ghi lại bộ ba số n, k, x . Trong đó x là số ở hệ cơ số b được chuyển đổi từ n . Ví dụ dưới đây minh họa cho file input và output của bài toán.

<u>Input.in</u>		<u>Output.out</u>		
5		8	2	1000
8	2	32	16	20
32	16	255	16	FF
255	16	100	10	100
100	10	64	32	20
64	32			

Ví dụ 2. Hãy viết chương trình tìm số các số tự nhiên N thỏa mãn đồng thời những điều kiện dưới đây ($N \leq 2^{31}$):

- N là số có K chữ số ($K \leq 15$).
- N là số nguyên tố.
- Đảo ngược các chữ số của N cũng là một số nguyên tố.
- Tổng các chữ số của N cũng là một số nguyên tố.
- Mỗi chữ số của N cũng là các số nguyên tố ;
- Thời gian thực hiện chương trình không quá 1sec.

Dữ liệu vào (Input) cho bởi file data.in theo khuôn dạng:

- Dòng đầu tiên ghi lại số tự nhiên M là số lượng các test ($M \leq 100$).
- M dòng kế tiếp ghi lại mỗi dòng một test. Mỗi test bao gồm một số K . Hai số được viết cách nhau một vài khoảng trắng.

Kết quả ra (Output): ghi lại M dòng trong file ketqua.out, mỗi dòng ghi lại bộ hai số N, X . Trong đó X là số các số có N chữ số thỏa mãn yêu cầu của bài toán. Ví dụ dưới đây minh họa cho file input và output của bài toán.

<u>Input.in</u>	<u>Output.out</u>
5	2 0
2	3 8
3	4 15
4	5 46
5	7 359
7	

Ví dụ. Cho cặp số tự nhiên N, K. Hãy xây dựng thuật toán tìm số các xâu nhị phân độ dài N không có dãy K số 1 liên tiếp.

Input:

- Dòng đầu tiên ghi lại số M là số lượng bộ test ($M \leq 100$).
- M dòng kế tiếp mỗi dòng ghi lại một bộ test. Mỗi bộ test bao gồm hai số N, K ($K < N \leq 50$).

Output: Ứng với mỗi bộ test đưa ra số các xâu nhị phân thỏa mãn yêu cầu bài toán.

Ví dụ về Input và Output của bài toán.

INPUT	OUTPUT
3	7
3 3	13
4 3	24
5 3	

1.6. Một số thuật toán cơ bản của logic & tập hợp

1.6.1. Định nghĩa và khái niệm

Định nghĩa 1. Cho $U = \{u_1, u_2, \dots, u_n\}$ là một tập hợp hữu hạn. Mỗi $u_i \in U$ được gọi là một thuộc tính. Tập U còn được gọi là tập thuộc tính.

Ví dụ 1. $U = \{\text{MASV}, \text{TSV}, \text{QUE}, \text{NS}\}$. Khi đó, MASV, TSV, QUE, NS được gọi là các thuộc tính. Ta hiểu, tập thuộc tính là tập thông tin chúng ta cần quản lý và xử lý.

Định nghĩa 2. Cho $U = \{u_1, u_2, \dots, u_n\}$ là một tập thuộc tính. Ứng với mỗi $u_i \in U$ ta làm tương ứng với một miền d_i được gọi là miền xác định của thuộc tính u_i . Tập $D = \{d_1 \times d_2 \times \dots \times d_n\}$ được gọi là miền xác định của thuộc tính U .

Ví dụ 2. $U = \{\text{MASV}, \text{TSV}, \text{QUE}, \text{NS}\}$ là tập thuộc tính. Ta làm tương ứng:

MASV : Tập số nguyên có 8 chữ số; QUE : Tập ký tự độ dài 50.

TSV : Tập ký tự độ dài 50; NS : Tập ký tự độ dài 8.

Định nghĩa 3. Quan hệ R với tập các thuộc tính $U = \{u_1, u_2, \dots, u_n\}$ là tập hợp các ánh xạ $t: U \rightarrow D$, sao cho hạn chế của ánh xạ t lên thuộc tính u_i , ký hiệu là $t(u_i) \in d_i$. Quan hệ R với tập thuộc tính là U được ký hiệu là $R(U)$, mỗi ánh xạ t được gọi là một bộ của quan hệ R .

$$R(U) = \{ t \mid t.u_i \in \text{dom}(u_i)\}$$

Ví dụ . $U = \{\text{MASV}, \text{TSV}, \text{QUE}, \text{NS}\}$. Khi đó, quan hệ $R(U)$ bao gồm các bộ t
được mô tả như trong Bảng 1 dưới đây.

MASV	TSV	QUE	NS
01	Nguyễn Tiến Thành	Hà Nội	1990
02	Trần Trung Hiếu	Hải Phòng	1991
03	Lê Bá Quyền	Hải Phòng	1992
04	Nguyễn Lan Anh	Hà Nội	1991
05	Lê Mậu Ngọ	Thái Bình	1993
06	Trần Tuấn Dũng	Thái Bình	1992
...			

1.6.2.Phụ thuộc dữ liệu:

Định nghĩa 1. Cho quan hệ $R(U)$, $X \subseteq U$, $Y \subseteq U$. Ta nói, tập thuộc tính X phụ thuộc hàm vào tập thuộc tính Y hay tập thuộc tính Y xác định hàm tập thuộc tính X và được ký hiệu là $f: X \rightarrow Y$ nếu với mọi bộ $t \in R$, $v \in R$ có $t.X = v.X$ thì cũng có $t.Y = v.Y$.

$$R(f: X \rightarrow Y) \Leftrightarrow \forall t \in R, v \in R \mid t.X = v.X \Rightarrow t.Y = v.Y.$$

Định nghĩa 2. Cho quan hệ $R(U)$, $X \subseteq U$, $Y \subseteq U$. Ta nói, quan hệ R thỏa phụ thuộc hàm $f: X \rightarrow Y$ khi và chỉ khi với mọi bộ t thuộc quan hệ R thì t thỏa mãn $f: X \rightarrow Y$. Ký hiệu quan hệ R thỏa phụ thuộc hàm f là $R(f)$.

$$R(f) \Leftrightarrow \forall t \in R \mid t(f)$$

Định nghĩa 3. Cho quan hệ $R(U)$, $X \subseteq U$, $Y \subseteq U$. Ta gọi $F = \{f: X \rightarrow Y \mid R(f)\}$ là tập phụ thuộc hàm trên U . Quan hệ R thỏa mãn tập phụ thuộc hàm F là $R(F)$.

$$R(F) \Leftrightarrow \forall f \in F \mid R(f)$$

1.6.3. Thuật toán tìm bao đóng

Cho quan hệ $R(U)$, F/U , $X \subseteq U$. Bài toán cơ bản đầu tiên là từ X ta có thể rút ra được những thông tin thuộc tính nào dựa trên F .

Định nghĩa 1. Ta gọi X^+ là tập thông tin được suy ra từ X dựa trên tập phụ thuộc hàm F được xác định theo thuật toán dưới đây là bao đóng của tập thuộc tính.

Thuật toán tìm Closure-Set:

Input :

- $U = \{ u_1, u_2, \dots, u_n \}$ là tập thuộc tính.
- F/U : Tập phụ thuộc hàm F trên U .
- $X \subseteq U$: X là tập con tập thuộc tính.

Output : $X^+ = \{ A \in U : F \rightarrow A \}$.

Actions:

- (i) Lấy $X^0 = X$;
- (ii) $X^i = X^{i-1} \cup W$ với $Z \subseteq X^0$ và $Z \rightarrow W \in F$ ($i=1, 2, \dots$).
- (iii) Nếu $X^i = X^{i-1}$ sau khi đã thử tất cả các phụ thuộc hàm thì $X^+ = X^i$.

End.

Ví dụ. Cho tập thuộc tính U , tập phụ thuộc hàm F/U và $X \subseteq U$ như dưới đây:

$$U = \{ ABCDEFGHIJ \};$$

$$F = \{ A \rightarrow BC; AB \rightarrow CD; E \rightarrow FG; EF \rightarrow GH; AE \rightarrow IJ \}.$$

$$X = AE$$

Hãy tìm $X^+ = ?$

Lời giải.

$$X^0 = X = AE;$$

$$X^1 = X^0 \cup BC = AEBC;$$

$$X^2 = X^1 \cup CD = AEBCD;$$

$$X^3 = X^2 \cup FG = AEBCDFG;$$

$$X^4 = X^3 \cup GH = AEBCDFGH;$$

$$X^5 = X^4 \cup IJ = AEBCDFGHIJ;$$

$$X^6 = X^5 = AEBCDFGHIJ = X^+.$$

Vậy $(AE)^+ = AEBCDFGHIJ$

1.6.4. Thuật toán tìm khóa

Trong các tập thuộc tính $K \subseteq U$ có $K^+ = U$. Ta có thể tìm ra được tập nhỏ nhất sao cho $K^+ = U$.

Định nghĩa 1. Ta gọi K là một siêu khóa nếu $K^+ = U$. Tập K được gọi là khóa nếu K là một siêu khóa và với mọi thuộc tính $u \in K$ thì $(K \setminus u)^+ \neq U$.

Thuật toán Key (U, F):

Input :

- U là tập thuộc tính.
- F là tập phụ thuộc hàm trên U .

Output: $K \subseteq U : K^+ = U \wedge \forall A \in K: (K \setminus A)^+ \neq U$.

Action:

$K = U; // K$ là một siêu khóa

for each $u_i \in K$ do

 if $(K \setminus u_i)^+ = U$ then

$K = (K \setminus u_i)$

 endif;

endfor;

Return(K);

End.

Ví dụ. Giả sử ta có tập thuộc tính $U = \{ABCDEFHIJ\}$ và tập phụ thuộc hàm $F = \{A \rightarrow BCD; E \rightarrow FGH; AE \rightarrow IJ\}$. Khi đó ta xác định khóa của $\langle U, F \rangle$ như sau :

- Chọn $K = U$ vì $K^+ = U^+ = U$ (đây là một siêu khóa).
- Lặp với các thuộc tính $u_i \in K$:
 - $u_1 = A \Rightarrow (K \setminus A)^+ = (BCDEFHIJ)^+ = BCDEFHIJ \neq U$.
 - $u_2 = B \Rightarrow (K \setminus B)^+ = (ACDEFHIJ)^+ = ABCDEFHIJ = U$; $K = ACDEFHIJ$.
 - $u_3 = C \Rightarrow (K \setminus C)^+ = (ADEFGHIJ)^+ = ABCDEFGHIJ = U$; $K = ADEFGHIJ$.
 - $u_4 = D \Rightarrow (K \setminus D)^+ = (AEFGHIJ)^+ = ABCDEFGHIJ = U$; $K = AEFGHIJ$.
 - $u_5 = E \Rightarrow (K \setminus E)^+ = (AFGHIJ)^+ = ABCDFGHIJ \neq U$.
 - $u_7 = F \Rightarrow (K \setminus F)^+ = (AEGHIJ)^+ = ABCDEFGHIJ = U$; $K = AEGHIJ$.
 - $u_8 = G \Rightarrow (K \setminus G)^+ = (AEHIJ)^+ = ABCDEFGHIJ = U$; $K = AEHIJ$.
 - $u_9 = H \Rightarrow (K \setminus H)^+ = (AEIJ)^+ = ABCDEFGHIJ = U$; $K = AEIJ$.
 - $u_{10} = I \Rightarrow (K \setminus I)^+ = (AEJ)^+ = ABCDEFGHIJ = U$; $K = AEJ$.
 - $u_{11} = J \Rightarrow (K \setminus J)^+ = (AE)^+ = ABCDEFGHIJ = U$; $K = AE$.

Từ đây ta kết luận $K = AE$ là khóa của $\langle U, F \rangle$.

1.6.5. Tính tương các tập phụ thuộc hàm

Định nghĩa 1. Hai tập phụ thuộc hàm F/U , G/U được gọi là tương đương nhau khi và chỉ khi

$$\begin{aligned} & \forall f \in F \mid G \rightarrow f \wedge \forall g \in G \mid F \rightarrow g \\ \Leftrightarrow & \forall f : X \rightarrow Y \in F \mid (X)_G^+ \supseteq Y \\ & \forall g : X \rightarrow Y \in G \mid (X)_F^+ \supseteq Y \end{aligned}$$

Thuật toán Img (F, G):

Input : F/U ; G/U ;

Output : - Return(True) nếu $F \Rightarrow G$;
- Return(False) Otherwise.

Actions :

```
for each g:X→Y in G do
    if ( Y ∉ (X)^+_F ) then
        Return(False);
    endif;
endfor;
return(True);
```

End.

Thuật toán Equivalence (F, G) {

```
    Return(Img( $F, G$ ) and Img( $G, F$ ));
}
```

Ví dụ. Giả sử ta có tập thuộc tính $U = \{A, B, C, D, E, F, G, H, I, J\}$, tập phụ thuộc hàm F và G dưới đây là tương đương.

$$F = \{ A \rightarrow BCD; E \rightarrow FGH; AE \rightarrow IJ \}$$

$$G = \{ A \rightarrow B; AB \rightarrow BCD; E \rightarrow F; EF \rightarrow FGH; AE \rightarrow IJ \}$$

Chứng minh. Ta cần chứng minh $\text{Img}(F, G) = \text{True}$ và $\text{Img}(G, F) = \text{True}$.

$f: X \rightarrow Y \in F$	$X^+_G = ?$	$X^+_G \supseteq Y ?$
$A \rightarrow BCD$	ABCD	Yes
$E \rightarrow FGH$	EFGH	Yes
$AE \rightarrow IJ$	ABCDEFGHIJ	Yes

$g: X \rightarrow Y \in G$	$X^+_F = ?$	$X^+_F \supseteq Y ?$
$A \rightarrow B$	ABCD	Yes
$AB \rightarrow BCD$	ABCD	Yes
$E \rightarrow F$	EFGH	Yes
$EF \rightarrow FGH$	EFGH	Yes
$AE \rightarrow IJ$	ABCDEFGHIJ	Yes

Định nghĩa 2. Tập phụ thuộc hàm G/U được gọi là phủ tự nhiên của F/U nếu G thỏa mãn ba điều kiện sau:

- (i) G tương đương với F ($G \sim F$).
- (ii) Với mọi $X \rightarrow Y \in G$ thì $X \cap Y = \emptyset$.
- (iii) Với mọi $X \rightarrow Y \in G$, $Z \rightarrow W \in G$ thì $X \neq Z$.

Thuật toán Natural_Cover(F):

Input : Tập phụ thuộc hàm F/U.

Output: G là phủ không dư của F.

Actions:

```
G = F; // G~F
for each g: X → Y ∈ G do
    G = G \ {X → Y} ∪ {X → Y\X};
endfor.
for each g: X → Y ∈ G do
    for each f: Z → W ∈ G do
        if (X = Z) then
            G = G \ {X → Y; Z → W} ∪ {X → YW};
        endif;
    endfor;
endfor.
Return(G);
```

End.

Ví dụ. Giả sử ta có tập thuộc tính U, tập phụ thuộc hàm F/U như dưới đây:

$$U = \{ABCDEFHIJ\},$$

$$F = \{ A \rightarrow B; AB \rightarrow BC; A \rightarrow D; E \rightarrow F; EF \rightarrow FG; F \rightarrow H; AE \rightarrow IJ \}$$

Khi đó phủ tự nhiên của F được xác định như sau.

Bước 1. G = F, khi đó:

$$G = \{ A \rightarrow B; AB \rightarrow BC; A \rightarrow D; E \rightarrow F; EF \rightarrow FG; F \rightarrow H; AE \rightarrow IJ \}$$

Bước 2 (vòng lặp for thứ nhất):

$$G = \{ A \rightarrow B; AB \rightarrow C; A \rightarrow D; E \rightarrow F; EF \rightarrow G; F \rightarrow H; AE \rightarrow IJ \}$$

Bước 3 (vòng lặp for thứ hai):

$$G = \{ A \rightarrow BD; AB \rightarrow C; E \rightarrow FH; EF \rightarrow G; F \rightarrow H; AE \rightarrow IJ \}$$

Từ đó ta xác định được G là phủ tự nhiên của F.

$$G = \{ A \rightarrow BD; AB \rightarrow C; E \rightarrow FH; EF \rightarrow G; F \rightarrow H; AE \rightarrow IJ \}$$

Định nghĩa 3. Tập phụ thuộc hàm F/U được gọi là tập không dư nếu với mọi phụ thuộc hàm $f: X \rightarrow Y \in F$ thì F không tương đương với $F \setminus \{X \rightarrow Y\}$. Tập phụ thuộc hàm G được gọi là phủ không dư của F nếu G tương đương với F và G là tập không dư.

Định lý. Phụ thuộc hàm $f: X \rightarrow Y \in F$ là không dư khi và chỉ khi $Y \not\subset X_{F \setminus f}^+$.

Thuật toán Non-Redundancy(F):

Input :

- F/U : Tập phụ thuộc hàm F trên U .

Output:

- G/U là phủ không dư của F .

Actions:

$G = F; // G$ tương đương với F

for each $g: X \rightarrow Y \in G$ do

if ($X_{G \setminus g}^+ \supseteq Y$) then

$G = G \setminus \{g: X \rightarrow Y\};$

endif;

endfor;

Return(G);

End.

Ví dụ. Giả sử ta có tập thuộc tính U, tập phụ thuộc hàm F/U như dưới đây:

$$U = \{ABCDEFHIJ\},$$

$$F = \{A \rightarrow BC; AB \rightarrow CD; AC \rightarrow BD; E \rightarrow F; EF \rightarrow GH; EG \rightarrow FH; AE \rightarrow IJ\}$$

Khi đó phủ tự không dư của F được xác định như sau:

- Bước 1: $G = F$, do vậy

$$G = \{A \rightarrow BC; AB \rightarrow CD; AC \rightarrow BD; E \rightarrow F; EF \rightarrow GH; EG \rightarrow FH; AE \rightarrow IJ\}$$

$g: X \rightarrow Y \in G$	$X^+_{G \setminus g} = ?$	$Y \subseteq X^+_{G \setminus g}$	$G = ?$
$A \rightarrow BC$	$(A)^+_{G \setminus \{A \rightarrow BC\}} = A$	No	
$AB \rightarrow CD$	$(AB)^+_{G \setminus \{AB \rightarrow CD\}} = ABCD$	Yes	$G = G \setminus g$
$AC \rightarrow BD$	$(AC)^+_{G \setminus \{AC \rightarrow BD\}} = AC$	No	
$E \rightarrow F$	$(E)^+_{G \setminus \{E \rightarrow F\}} = E$	No	
$EF \rightarrow GH$	$(EF)^+_{G \setminus \{EF \rightarrow GH\}} = EF$	No	
$EG \rightarrow FH$	$(EG)^+_{G \setminus \{EG \rightarrow FH\}} = EFGH$	Yes	$G = G \setminus g$
$AE \rightarrow IJ$	$(AE)^+_{G \setminus \{AE \rightarrow IJ\}} = ABCDEFGHEFGH$	No	
$G = \{A \rightarrow BC; AC \rightarrow BD; E \rightarrow F; EF \rightarrow GH; AE \rightarrow IJ\}$			

Định nghĩa 4. Cho phụ thuộc hàm $f: X \rightarrow Y \in F$. Thuộc tính $A \in X$ được gọi là thuộc tính dư bên phải nếu F tương đương với $F \setminus \{X \setminus A \rightarrow Y\}$. Thuộc tính $A \in Y$ được gọi là thuộc tính dư bên phải nếu F tương đương với $F \setminus \{X \rightarrow Y \setminus A\}$. Tập phụ thuộc hàm G được gọi là phủ thu gọn của F nếu F là tập không dư và tất cả $f : X \rightarrow Y \in F$ đều không có thuộc tính dư trái và thuộc tính dư phải.

Định lý. Thuộc tính $A \in X$ là thuộc tính dư trái của phụ thuộc hàm $f: X \rightarrow Y$ khi và chỉ khi $A \in (X \setminus A)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \setminus A \rightarrow Y\}}$. Thuộc tính $A \in Y$ là thuộc tính dư phải của phụ thuộc hàm $f: X \rightarrow Y$ khi và chỉ khi $Y \subseteq (X)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus A\}}$.

Định nghĩa 5. Tập phụ thuộc hàm G/U được gọi là phủ thu gọn của F/U nếu G là phủ của F và với mọi phụ thuộc hàm $g: X \rightarrow Y \in G$ thì g không chứa thuộc tính dư trái và không chứa thuộc tính dư phải.

Thuật toán Left-Cut ($f:X \rightarrow Y$): //Làm gọn về trái của $f:X \rightarrow Y$.

Input :

- F/U : Tập phụ thuộc hàm F trên U .
- $f:X \rightarrow Y \in F$: phụ thuộc hàm f .

Output:

- $f':X' \rightarrow Y'$ và f không chứa thuộc tính dư trái.

Format : $f = \text{Left-Cut}(X \rightarrow Y)$;

Actions:

```
for each attribute  $A \in X$  do {
    if ( $A \in (X \setminus A)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \setminus A \rightarrow Y\}}$ ) then
         $X = X \setminus A;$ 
    endif;
}
endfor;
return( $f:X \rightarrow Y$ );
```

End.

Ví dụ. Giả sử ta có tập thuộc tính U, tập phụ thuộc hàm F/U như dưới đây:

$$U = \{ABCDEFHIJ\},$$

$$F = \{A \rightarrow BC; AC \rightarrow BD; E \rightarrow F; EF \rightarrow GH; AE \rightarrow IJ\}$$

$$f: AC \rightarrow BD \in F \text{ và } f: EF \rightarrow GH$$

Khi đó thủ tục Left-Cut(AC->BD) được thực hiện như sau.

$A \in X$	$(X \setminus A)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \setminus A \rightarrow Y\}} = ?$	$A \in (X \setminus A)^+$	$X = ?$
$A \in AC$	$(AC \setminus A)^+_{F \setminus \{AC \rightarrow BD\} \cup \{C \rightarrow BD\}} = C$	No	AC
$C \in AC$	$(AC \setminus C)^+_{F \setminus \{AC \rightarrow BD\} \cup \{A \rightarrow BD\}} = ABCD$	Yes	A
$f: A \rightarrow BD$			

$A \in X$	$(X \setminus A)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \setminus A \rightarrow Y\}} = ?$	$A \in (X \setminus A)^+$	$X = ?$
$E \in EF$	$(EF \setminus E)^+_{F \setminus \{EF \rightarrow GH\} \cup \{F \rightarrow GH\}} = F$	No	EF
$F \in EF$	$(EF \setminus F)^+_{F \setminus \{EF \rightarrow GH\} \cup \{E \rightarrow GH\}} = EFGH$	Yes	E
$f: E \rightarrow EF$			

Thuật toán Right-Cut (f:X->Y): //Làm gọn về phải của f:X->Y.

Input :

- F/U : Tập phụ thuộc hàm F trên U.
- $f:X \rightarrow Y \in F$: phụ thuộc hàm f.

Output:

- $f':X' \rightarrow Y'$ và f không chứa thuộc tính dư phải.

Format : $f = \text{Right-Cut}(X \rightarrow Y);$

Actions:

```
for each attribute A ∈ Y do {  
    if  $((X)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus A\}} \supseteq Y$  then  
         $Y = Y \setminus A;$   
    endif;  
endfor;  
return(f:X → Y);
```

End.

Ví dụ. Giả sử ta có tập thuộc tính U, tập phụ thuộc hàm F/U như dưới đây:

$$U = \{ABCDEFHIJ\},$$

$$F = \{A \rightarrow BC; AC \rightarrow BD; E \rightarrow FH; EF \rightarrow GH; AE \rightarrow IJ\}$$

$$f: AC \rightarrow BD \in F \text{ và } f: EF \rightarrow GH$$

Khi đó thủ tục Left-Cut(AC->BD) được thực hiện như sau.

$A \in Y$	$(X)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus A\}} = ?$	$Y \subseteq (X)^+$	$Y = ?$
$B \in BD$	$(AC)^+_{F \setminus \{AC \rightarrow BD\} \cup \{AC \rightarrow BD \setminus B\}} = ABCD$	Yes	D
$D \in D$	$(AC)^+_{F \setminus \{AC \rightarrow D\} \cup \{A \rightarrow \emptyset\}} = ABC$	No	D
f: AC → D			

$A \in Y$	$(X)^+_{F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus A\}} = ?$	$Y \subseteq (X)^+$	$Y = ?$
$G \in GH$	$(EF)^+_{F \setminus \{EF \rightarrow GH\} \cup \{EF \rightarrow H\}} = EFH$	No	GH
$H \in GH$	$(EF)^+_{F \setminus \{EF \rightarrow GH\} \cup \{EF \rightarrow G\}} = EFGH$	Yes	G
f: EF → G			

Thuật toán Optimization-Cover (F): // Tìm phủ thu gọn của F.

Input :

- U là tập thuộc tính.
- F/U : Tập phụ thuộc hàm F trên U.

Output: G là phủ thu gọn của F.

Format : G = Optimal_Cover (F);

Actions:

$G = \emptyset;$

for each $f: X \rightarrow Y \in F$ do {

$f = \text{Left-Cut}(f); f = \text{Right-Cut}(F);$

$G = G \cup f;$

endfor;

return(G);

End.

Ví dụ. Giả sử ta có tập thuộc tính U, tập phụ thuộc hàm F/U như dưới đây:

$$U = \{ABCDEFHIJ\},$$

$$F = \{A \rightarrow BC; AC \rightarrow BD; E \rightarrow FH; EF \rightarrow GH; AE \rightarrow IJ\}$$

Khi đó thủ tục Left-Cut(AC->BD) được thực hiện như sau.

$f:X \rightarrow Y \in F$	$f = \text{Left-Cut}(X \rightarrow Y)$	$f = \text{Right-Cut}(X \rightarrow Y)$	Kết quả
$A \rightarrow BC$	$A \rightarrow BC$	$A \rightarrow BC$	$A \rightarrow BC$
$AC \rightarrow BD$	$A \rightarrow BD$	$A \rightarrow D$	$A \rightarrow D$
$E \rightarrow FH$	$E \rightarrow FH$	$E \rightarrow FH$	$E \rightarrow FH$
$EF \rightarrow GH$	$E \rightarrow GH$	$E \rightarrow H$	$E \rightarrow H$
$AE \rightarrow IJ$	$AE \rightarrow IJ$	$AE \rightarrow IJ$	$AE \rightarrow IJ$
$G = \{A \rightarrow BC; A \rightarrow D; E \rightarrow FH; AE \rightarrow IJ\}$			

1.7. CASE STUDY 1:

1. Lập trình theo các thuật toán đã trình bày ở trên bằng C++, Java?
2. Xác định độ phức tạp các thuật toán đã được nêu ở trên?
3. Xu hướng nghiên cứu, phát triển cho các thuật toán trên ?
4. Hãy mô tả, tính toán độ phức tạp thuật toán và cài đặt những thuật toán dưới đây:
 - Thuật toán DFS, BFS, Kruskal, Prim, Dijkstra, Floyed, Euler-Cycle, Hamilton-Cycle?
 - Thuật toán Selection-Sort, Insertion-Sort, Bubble-Sort, Quick-Sort, Merge-Sort, Heap-Sort, Radix-Sort, Sequential-Search, Binary-Search.