Chương 2: Cơ sở phân tích độ phức tạp giải thuật

Phân tích giải thuật là sự khảo sát tính *hiệu quả* của một giải thuật dưới hai tiêu chuẩn được quan tâm nhiều nhất: thời gian (thực hiện) và không gian (tài nguyên chiếm dụng).

Kích thước dữ liệu nhập

Gọi n là kích thước dữ liệu. Khi đó, thì độ phức tạp (hay tính hiệu quả) là một hàm theo n: $f(n)$.

Đơn vị trong phép đo thời gian thực thi

Sử dụng thời gian vật lý (giờ, phút, giây, …) không phải là lựa chọn tốt.

Độ hiệu quả về mặt thời gian của một giải thuật sẽ được xác định dựa trên việc đếm số lần thực hiện của thao tác cơ sở với dữ liệu đầu vào có kích thước n.
Chú ý: Thao tác cơ sở là những thao tác mà sự thực thi của nó đóng góp chủ yếu vào thời gian thực thi chung của giải thuật.

Để *ước lượng* thời gian vật lý $T(n)$ cho một giải thuật, ta có thể sử dụng công thức:

$$T(n) \approx t \times f(n)$$

với t là thời gian thao tác cơ sở thực thi trên một phần cứng cụ thể, $f(n)$ là số lần thao tác cơ sở phải thực hiện và n là kích thước dữ liệu nhập.

Sự định lượng tính hiệu quả của một giải thuật chỉ dừng ở mức xấp xỉ.

Tốc độ hay bậc tăng trưởng

Khi n đủ lớn, các hằng số cũng như những toán hạng có bậc nhỏ của đa thức $f(n)$ không còn mang nhiều ý nghĩa nên được loại bỏ.

Ví dụ: Xét hai hàm $0.1n^2 + n + 100$ và $0.1n^2$.

$\boldsymbol{n}$	$\mathbf{0.1}\boldsymbol{n}^{\mathbf{2}}$	$\mathbf{0.1}\boldsymbol{n}^{\mathbf{2}} + \boldsymbol{n} + \mathbf{100}$
10	10	120
100	1000	1200
1000	100000	101100

Nếu $f(n) = \frac{1}{2}n(n-1)$ thì khi n đủ lớn, $f(n) = \frac{1}{2}n^2 - \frac{1}{2}n \approx n^2$.

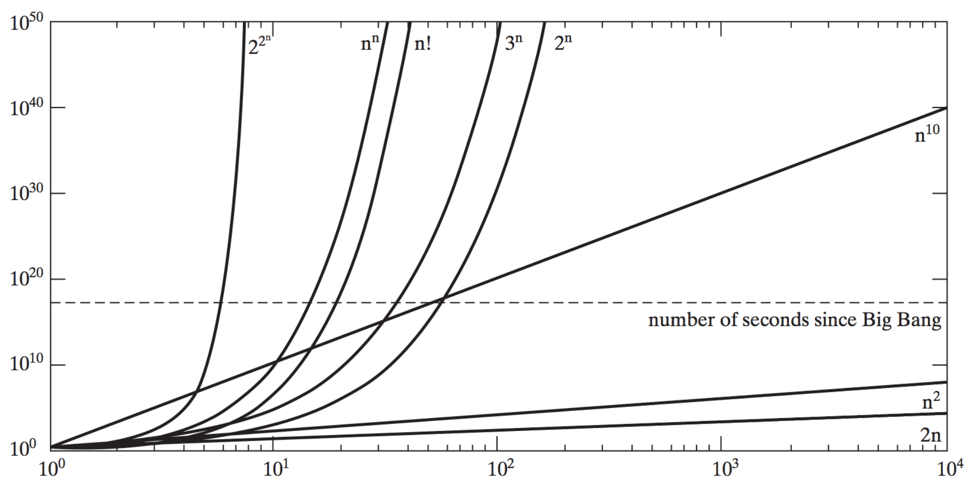
Nếu giải thuật này phải xử lý lượng dữ liệu lớn gấp hai lần thì thời gian sẽ tăng lên bao nhiêu lần?

$$\frac{T(2n)}{T(n)} = \frac{t \times f(2n)}{t \times f(n)} = \frac{(2n)^2}{n^2} = 4$$

Giá trị 4 thể hiện mức "tăng trưởng" của giải thuật và chú ý là t không còn cần thiết trong phép tính (trừu tượng hóa hoàn toàn, không phụ thuộc vào phần cứng cụ thể).

Bảng dưới đây chứa giá trị của một số hàm chuẩn quan trọng, được dùng để biểu diễn tính hiệu quả của giải thuật.

n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10^1	3.3	10^1	3.3×10^1	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
10^3	10	10^3	10×10^3	10^6	10^9		
10^4	13	10^4	13×10^4	10^8	10^{12}		
10^5	17	10^5	17×10^5	10^{10}	10^{15}		
10^6	20	10^6	20×10^6	10^{12}	10^{18}		



Hàm lũy thừa 2^n, hàm giai thừa $n!$ và hàm luỹ thừa n (n^n) có sự tăng trưởng khủng khiếp. Những giải thuật loại này chỉ có thể chạy với kích thước dữ liệu nhỏ.

Trường hợp tốt nhất, trung bình và xấu nhất

Với nhiều giải thuật, tính hiệu quả của chúng không chỉ phụ thuộc vào kích thước dữ liệu nhập mà còn là đặc trưng/phân bố của mỗi dữ liệu đầu vào.

Ví dụ: Giải thuật tìm tuần tự giá trị k trong dãy n giá trị.

Trường hợp tốt nhất (*best-case*) $B(n)$ là hàm chỉ ra số lần thực thi ít nhất của thao tác cơ sở mà giải thuật phải tiến hành trên bất kỳ thể hiện dữ liệu nào có kích thước n.

Trường hợp trung bình (*average-case*) $A(n)$ là hàm chỉ ra số lần thực thi trung bình của thao tác cơ sở mà giải thuật phải tiến hành trên bất kỳ thể hiện dữ liệu nào có kích thước n.

Trường hợp xấu nhất (*worst-case*) $W(n)$ là hàm chỉ ra số lần thực thi nhiều nhất của thao tác cơ sở mà giải thuật phải tiến hành trên bất kỳ thể hiện dữ liệu nào có kích thước n.

Ví dụ: Tìm giá trị k trong dãy n giá trị.

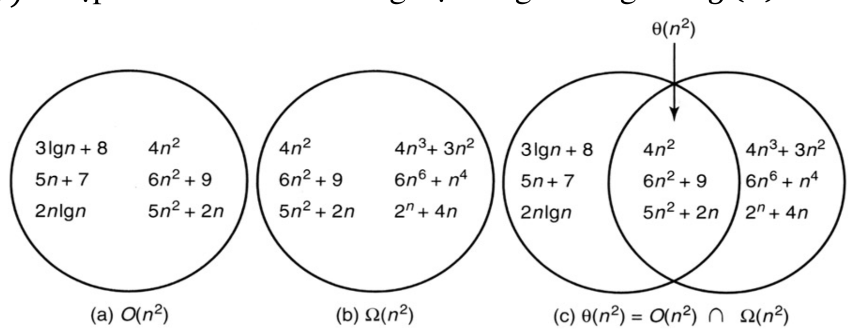
```
search(a[1 .. n], k) {
   for (i = 1; i ≤ n; i++)
      if (a[i] == k)
         return   1;
   return   0;
}
```

$B(n) = 1$
$W(n) = n$
$A(n) = ?$

Các ký hiệu tiệm cận

Gọi $f(n)$ và $g(n)$ là hai hàm dương, định nghĩa trên $\mathbb{N}$. Hàm $f(n)$ để chỉ "thời gian" thực thi của giải thuật, còn $g(n)$ là một hàm nào đó dùng để so sánh.

- $O(g(n))$ là tập của các hàm có bậc tăng trưởng nhỏ hơn hoặc bằng $g(n)$.
- $\Omega(g(n))$ là tập của các hàm có bậc tăng trưởng lớn hơn hoặc bằng $g(n)$.
- $\Theta(g(n))$ là tập của các hàm có cùng bậc tăng trưởng như $g(n)$.



Ký hiệu O

Định nghĩa: Cho hàm $g(n)$. Ký hiệu $O\big(g(n)\big)$ là tập của các hàm:

$$O\big(g(n)\big) = \{f(n){:}\, \exists c \in \mathbb{R}^+ \wedge n_0 \in \mathbb{N}, 0 \le f(n) \le cg(n), \forall n \ge n_0\}$$

Ví dụ: Gọi $f(n) = n^2 + 2n + 1$, $g(n) = n^2$ thì $f(n) \in O(n^2)$

Ký hiệu Ω

Định nghĩa: Cho hàm $g(n)$. Ký hiệu $\Omega\big(g(n)\big)$ là tập của các hàm:

$$\Omega\big(g(n)\big) = \{f(n){:}\, \exists c \in \mathbb{R}^+ \wedge n_0 \in \mathbb{N}, 0 \le cg(n) \le f(n), \forall n \ge n_0\}$$

Ví dụ: Nếu $f(n) = n^2 + 2n$, $g(n) = n^2$ thì $f(n) \in \Omega(n^2)$.

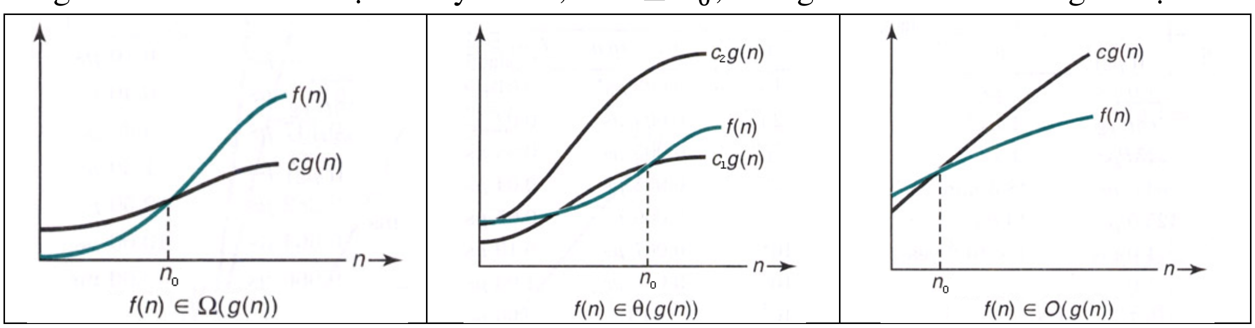
Ký hiệu Θ

Định nghĩa: Cho hàm $g(n)$. Ký hiệu $\Theta\big(g(n)\big)$ là tập của các hàm:

$$\Theta\big(g(n)\big) = \{f(n){:}\, \exists c_1, c_2 \in \mathbb{R}^+ \wedge n_0 \in \mathbb{N}, 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0\}$$

Ví dụ: Nếu $f(n) = \frac{1}{2}n^2 - 3n$, $g(n) = n^2$ thì $f(n) \in \Theta(n^2)$

Dưới đây là đồ thị, biểu diễn ý nghĩa của ký hiệu tiệm cận. Khi $n < n_0$, các đường cong biểu diễn chưa ổn định. Tuy nhiên, từ $n \ge n_0$, chúng tuân thủ các biên giới hạn.



Một số định lý liên quan

Định lý 1: Với hai hàm giá trị dương bất kỳ $f(n)$ và $g(n)$:

$$f(n) \in \Theta\big(g(n)\big) \Leftrightarrow f(n) \in O\big(g(n)\big) \wedge f(n) \in \Omega\big(g(n)\big)$$

Định lý 2: Cho đa thức $f(n) = \sum_{i=0}^{d} a_i n^i$ với $a_d > 0$:

$$f(n) \in O(n^d)$$

với $c = \sum_{i=0}^{d} |a_i|$, $\forall n > 1$.

Định lý 3: Nếu $f_1(n) \in O\big(g_1(n)\big)$ và $f_2(n) \in O\big(g_2(n)\big)$:

$$f_1(n) + f_2(n) \in O\Big(max\big(g_1(n), g_2(n)\big)\Big)$$

Ví dụ: Kiểm tra dãy số (chiều dài *n*) có chứa hai số giống nhau không?

Phân tích giải thuật không đệ qui

Sự phân tích thường trải qua những bước sau:

1. Xác định kích thước dữ liệu nhập.
2. Nhận diện (các) thao tác cơ sở.
3. Kiểm tra xem số lần thực hiện của thao tác cơ sở có phụ thuộc vào sự phân bố của dữ liệu nhập hay không? Nếu có thì phải xác định trường hợp tốt nhất, xấu nhất và nếu có thể là trung bình.
4. Xây dựng đa thức $f(n)$ mô tả tổng số lần thao tác cơ sở được thực thi.
5. Qui đa thức $f(n)$ về hàm chuẩn biểu diễn bậc tăng trưởng tương ứng.

Ví dụ: Tìm số lớn nhất trong dãy n số.

```
MaxElement(a[1 .. n]) {
   max = a[1];
   for (i = 2; i ≤ n; i++)
      if (a[i] > max)
         max = a[i];
   return   max;
}
```

Ví dụ: Nhân hai ma trận $n \times n$.

```
MatrixMultiplication(a[1 .. n, 1 .. n], b[1 .. n, 1 .. n]) {
   for (i = 1; i ≤ n; i++)
      for (j = 1; j ≤ n; j++) {
         c[i, j] = 0;
         for (k = 1; k ≤ n; k++)
            c[i, j] = c[i, j] + a[i, k] * b[k, j];
      }
}
```

Ví dụ: Sắp xếp nổi bọt

```
BubbleSort(a[1 .. n]) {
   for (i = 2; i < n; i++)
      for (j = n; j ≥ i; j--)
         if (a[j - 1] > a[j])
            a[j - 1] ⇆ a[j];
}
```

Gọi $C(n)$ là số lần thực hiện phép so sánh với kích thước dữ liệu n:

$$C(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Cải tiến

```
BubbleSort(a[1 .. n]) {
   flag = true;
   m = 1;
   while (flag) {
      flag = false;
      m++;
      for (j = n; j ≥ m; j--)
         if (a[j - 1] > a[j]) {
            a[j - 1] ⇆ a[j];
            flag = true;
         }
   }
}
```

Trường hợp tốt nhất: $B(n) = (n - 1)$

Trường hợp xấu nhất:

$$W(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Trường hợp trung bình:

Gọi $C(i)$ là tổng số lần so sánh sau lượt duyệt thứ i. Vì chúng ta có $(n - 1)$ khả năng dừng nên chi phí trung bình là:

$$A(n) = \frac{1}{n-1}\sum_{i=1}^{n-1} C(i) \in \Theta(n^2)$$

Ví dụ: Sắp xếp Chèn

<table>
<tr><th>Phiên bản gốc</th><th>Sử dụng lính canh</th></tr>
<tr><td>

```
InsertionSort(a[1 .. n]) {
   for (i = 2; i ≤ n; i++) {
      v = a[i];
      j = i - 1;
      while (j ≥ 1) && (a[j] > v) {
         a[j + 1] = a[j];
         j--;
      }
      a[j + 1] = v;
   }
}
```

</td><td>

```
InsertionSortwithSentinel(a[1 .. n])
{
   for (i = 2; i ≤ n; i++) {
      a[0] = v = a[i];
      j = i - 1;
      while (a[j] > v) {
         a[j + 1] = a[j];
         j--;
      }
      a[j + 1] = v;
   }
}
```

</td></tr>
</table>

Trường hợp tốt nhất: $B(n) = n - 1 \in \Theta(n)$

Trường hợp xấu nhất:

$$W(n) = \sum_{i=2}^{n}(i-1) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

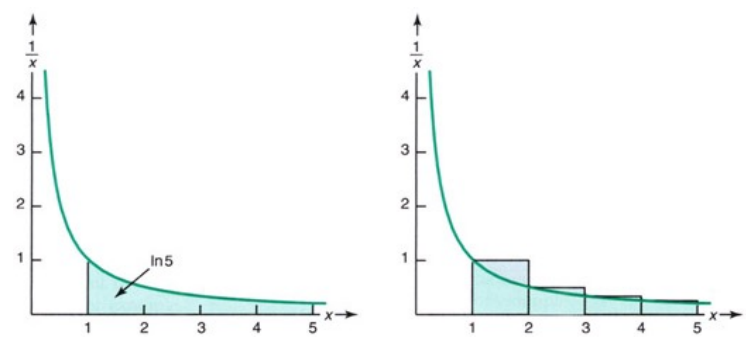
Trường hợp trung bình: Gọi $C(i)$ là số lần so sánh trung bình khi xét vị trí i.

$$C(i) = \frac{1}{i} \times (i-1) + \sum_{j=1}^{i-1} \frac{1}{i} \times j$$

Từ đây, số lượng trung bình của các phép so sánh $A(n)$ là:

$$A(n) = \sum_{i=2}^{n} C(i) \approx \frac{n^2 - n}{4} + n - \ln n - \gamma \approx \frac{n^2}{4} \in \Theta(n^2)$$

Gợi ý:



$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln n + \gamma$$

với $\gamma = 0.5772 \ldots$ là hằng số Euler.

Ví dụ: Số lượng bit trong biểu diễn nhị phân của số nguyên dương n?

```
Binary(n) {
   count = 1;
   while (n > 1) {
      count++;
      n = ⌊n / 2⌋;
   }
   return   count;
}
```

Công thức chính xác để tính số lượng bit là: $\lfloor \log_2 n \rfloor + 1 \in \Theta(\log n)$

Hệ thức truy hồi (Recurrence relations)

Ví dụ: Có bao nhiêu chuỗi nhị phân chiều dài n không chứa hai ký số 0 liên tiếp?

Ví dụ: Họ nhà thỏ và dãy số Fibonacci (Fibonacci, 1202)

Một cặp thỏ mới sinh (1 đực, 1 cái) được thả lên hoang đảo. Giả sử chúng trường thọ và độ tuổi sinh sản là *sau* 2 tháng tuổi. Khi đạt đến tuổi sinh sản thì hàng tháng, thỏ cái sẽ sinh ra một cặp đủ cả nếp lẫn tẻ. Cho biết số *cặp* thỏ trên đảo trong tháng thứ n?

Đầu tiên, vì là hoang đảo nên ban đầu có 0 cặp thỏ (F_0).

Tuổi > 2	Tuổi ≤ 2	(trong) Tháng	Tổng cặp
	①	1	1
	②	2	1
③	①	3	2
④	② ①	4	3
⑤ ③	② ① ①	5	5
⑥ ④ ③	② ② ① ① ①	6	8

Đến tháng thứ n (F_n): trên đảo có $F_{n-1} + F_{n-2}$ cặp, gồm F_{n-2} cặp mới sinh và F_{n-1} cặp đã có trên đảo ở tháng trước đó. Công thức tổng quát là:

$$F_n = F_{n-1} + F_{n-2} \text{ với } F_0 = 0, F_1 = 1$$

Các phương pháp giải hệ thức truy hồi

Tìm lời giải cho một hệ thức truy hồi với ràng buộc của điều kiện đầu nghĩa là tìm công thức tường minh biểu diễn toán hạng chung (hoặc chứng minh rằng một dãy như thế không tồn tại).

Phương pháp thay thế tiến (*Method of forward substitutions*)

Sử dụng điều kiện đầu của dãy, thông qua hệ thức truy hồi, xác định giá trị của một số toán hạng đứng đầu dãy. Từ đây, cố gắng suy ra công thức để tính toán hạng chung.

Ví dụ: Xét hệ thức truy hồi và điều kiện đầu sau

$$x_n = 2x_{n-1} + 1 \text{ và } x_1 = 1$$

Nhận thấy:

$$x_1 = 1$$
$$x_2 = 2x_1 + 1 = 3$$
$$x_3 = 2x_2 + 1 = 7$$
$$x_4 = 2x_3 + 1 = 15$$
$$\dots$$

Công thức đề nghị là: $x_n = 2^n - 1$ với $n > 0$

Phương pháp thay thế lùi (*Method of backward substitutions*)

Xây dựng x_n như là một hàm của x_{n-i} với $i = 1,2, \dots$

Chọn giá trị i sao cho $n - i$ đạt đến điều kiện đầu. Từ đây, có thể suy ra được công thức hay lời giải của hệ thức truy hồi.

Ví dụ: $x_n = x_{n-1} + n$ và $x_0 = 0$

Hệ thức truy hồi tuyến tính thuần nhất bậc k với các hệ số hằng

$$x_n = c_1 x_{n-1} + c_2 x_{n-2} + \cdots + c_k x_{n-k}$$

hay

$$f(n) = x_n - c_1 x_{n-1} - c_2 x_{n-2} - \cdots - c_k x_{n-k} = 0$$

với $c_1, c_2, \dots, c_k \in \mathbb{R}$, $c_k \neq 0$ và hệ thức có k điều kiện đầu là

$$x_0 = C_0, x_1 = C_1, \dots, x_{k-1} = C_{k-1}$$

Điều này cho thấy, hệ thức có vô số lời giải (vì phụ thuộc vào giá trị của điều kiện đầu).

Lời giải (hay công thức tường minh) của hệ thức sẽ có dạng $x_n = r^n$ với r là một hằng số. Hay nói cách khác, nếu $x_n = r^n$ là một lời giải thì:

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k}$$

Chia hai vế cho r^{n-k} và chuyển vế, ta được:

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_k r^0 = 0$$

Đây được gọi là *phương trình đặc trưng* (*characteristic equation*) của hệ thức truy hồi. Dãy $\{x_n\}$ có lời giải $x_n = r^n$ nếu và chỉ nếu r là nghiệm của phương trình này.

Hệ thức truy hồi tuyến tính thuần nhất bậc 2 với các hệ số hằng

Phương trình đặc trưng trong trường hợp này có dạng:

$$ar^2 + br + c = 0$$

Gọi r_1 và r_2 là nghiệm của phương trình này.

Trường hợp 1: Nếu r_1 và r_2 là số thực, $r_1 \neq r_2$. Lời giải tổng quát là:

$$x_n = \alpha {r_1}^n + \beta {r_2}^n, \forall \alpha, \beta \in \mathbb{R}$$

Trường hợp 2: Nếu $r_1 = r_2$ là số thực. Lời giải tổng quát là:

$$x_n = \alpha r^n + \beta n r^n, \forall \alpha, \beta \in \mathbb{R}$$

Trường hợp 3: Nếu $r_{1,2} = u \pm iv$ thì lời giải là:

$$x_n = \gamma^n(\alpha \cos n\theta + \beta \sin n\theta)$$

$\forall \alpha, \beta \in \mathbb{R}, \gamma = \sqrt{u^2 + v^2}, \theta = arctan(v/u)$.

Ví dụ: Xét hệ thức truy hồi $x_n = x_{n-1} + 2x_{n-2}$ với $x_0 = 2, x_1 = 7$.
Lời giải của dãy $\{x_n\}$ là:

$$x_n = 3 \times 2^n - (-1)^n$$

Ví dụ: Xét hệ thức truy hồi $x_n = 6x_{n-1} - 9x_{n-2}$ với $x_0 = 0, x_1 = 3$
Lời giải là:

$$x_n = n3^n$$

Phân tích giải thuật đệ qui

Phương pháp phân tích

Đối với các giải thuật đệ qui, quá trình phân tích *thường* diễn ra như sau:

1. Xác định kích thước dữ liệu nhập.
2. Nhận diện (các) thao tác cơ sở.
3. Kiểm tra xem số lần thực hiện của thao tác cơ sở có biến đổi trên những thể hiện dữ liệu nhập cùng kích thước hay không? Nếu có thì cần xác định thêm trường hợp tốt nhất, xấu nhất và (nếu có thể) là trung bình.
4. Xây dựng *hệ thức truy hồi* cùng *điều kiện đầu* nhằm mô tả số lần thực hiện của thao tác cơ sở.
5. Giải hệ thức truy hồi và qui về hàm chuẩn biểu diễn bậc tăng trưởng tương ứng.

Ví dụ: Tính n!

```
Factorial(n) {
   if (n == 0)
      return   1;
   return   Factorial(n - 1) * n;
}
```

Hệ thức truy hồi là:

$$M(n) = M(n-1) + 1 \text{ với } M(0) = 0$$

Dùng *phương pháp thay thế lùi*, ta có được: $M(n) \in \Theta(n)$

Ví dụ: Tháp Hà nội

```
HNTower(n, left, middle, right) {
   if (n) {
      HNTower(n - 1, left, right, middle);
      Movedisk(1, left, right);
      HNTower(n - 1, middle, left, right);
   }
}
```

Kích thước dữ liệu là n. Gọi số lần chuyển dời n đĩa là $M(n)$. Hệ thức truy hồi là:

$$M(n) = M(n-1) + 1 + M(n-1) = 2M(n-1) + 1$$

$$M(1) = 1$$

Bằng phép thay thế lùi, ta có $M(n) = 2^n - 1 \in \Theta(2^n)$

Ví dụ: Số lượng bit trong biểu diễn nhị phân của số nguyên dương n?

```
BitCount(n) {
   if (n == 1) return   1;
   return   BitCount(⌊n / 2⌋) + 1;
}
```

Thao tác cơ sở chính là phép cộng. Gọi $A(n)$ là số lượng phép cộng cần phải thực thi với kích thước dữ liệu n. Khi đó, $A\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$ là số lượng phép cộng cần phải thực thi với kích thước dữ liệu $\left\lfloor\frac{n}{2}\right\rfloor$. Ta có hệ thức truy hồi như sau:

$$A(n) = A\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1 \text{ và } A(1) = 0$$

vì khi đó không thực hiện phép cộng nào cả.

Định lý "*smoothness rule*":

"*Bậc tăng trưởng* quan sát được đối với trường hợp n là lũy thừa của 2 cũng sẽ là *bậc tăng trưởng* của mọi giá trị có thể có của n".

Giả sử $n = 2^k$, hệ thức truy hồi trở thành:

$$A(2^k) = A(2^{k-1}) + 1$$

$$A(2^0) = 0$$

Tiến trình thay thế lùi cho thấy:

$$A(n) = \log_2 n \in \Theta(\log n)$$

Ví dụ: Họ nhà thỏ và dãy số Fibonacci.

Hệ thức truy hồi được viết lại là:

$$F_n - F_{n-1} - F_{n-2} = 0$$

Phương trình đặc trưng là:

$$r^2 - r - 1 = 0$$

với nghiệm:

$$r_{1,2} = \frac{1 \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2} = \frac{1 \pm \sqrt{5}}{2}$$

Ta có:

$$F_n = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Để tìm giá trị của α và β, ta dựa vào hệ phương trình sau:

$$F_0 = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^0 + \beta\left(\frac{1-\sqrt{5}}{2}\right)^0 = 0$$

$$F_1 = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^1 + \beta\left(\frac{1-\sqrt{5}}{2}\right)^1 = 1$$

Giải hệ phương trình này, ta có được $\alpha = 1/\sqrt{5}$ và $\beta = -1/\sqrt{5}$. Vậy, F_n là:

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Gọi $\phi = (1+\sqrt{5})/2 \approx 1.61803$, $\hat{\phi} = \left(1-\sqrt{5}\right)/2 = -1/\phi \approx -0.61803$:

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

Vì $-1 < \hat{\phi} < 0$ nên $\hat{\phi}^n$ sẽ rất nhỏ khi $n \to \infty$. Từ đây, để đơn giản, người ta tính $F_n = \frac{1}{\sqrt{5}}\phi^n$ rồi làm tròn về số nguyên gần nhất.

Các giải thuật tính F_n

Đệ qui

```
Fibonacci(n) {
   if (n ≤ 1)
      return   n;
   return   Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

Gọi $A(n)$ là số phép cộng phải thực hiện để tìm số F_n. Vì mỗi lần hàm được gọi thì phép cộng thực thi một lần nên hệ thức truy hồi tính $A(n)$ là:

$$A(n) = A(n-1) + A(n-2) + 1 \text{ với } n > 1$$

và điều kiện đầu là $A(0) = 0$, $A(1) = 0$ (vì nếu $n \leq 1$ thì không thực hiệp phép cộng). Giải hệ thức, ta có:

$$A(n) = \frac{1}{\sqrt{5}}\left(\phi^{n+1} - \hat{\phi}^{n+1}\right) - 1 \in \Theta(\phi^n)$$

Không đệ qui

Dựa vào công thức $F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$, cài đặt với chi phí tuyến tính.

Qui hoạch động:

```
Fibonacci(n) {
   if (n ≤ 1)
      return   n;
   f0 = 0; f1 = 1;
   for (i = 2; i ≤ n; i++) {
      fn = f0 + f1;
      f0 = f1;
      f1 = fn;
   }
   return   fn;
}
```

```
Fibonacci(n) {
   if (n ≤ 1)
      return   n;
   f0 = 0;
   f1 = 1;
   for (i = 2; i ≤ n; i++) {
      f1 = f1 + f0;
      f0 = f1 - f0;
   }
   return f1;
}
```

Giải thuật có chi phí tuyến tính.

Lũy thừa ma trận

Phương pháp này có chi phí $\Theta(\log n)$, sử dụng đẳng thức sau:

$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \text{ với } n \geq 1$$

Nhận thấy, để tính lũy thừa n của ma trận $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, ta sẽ tính như sau:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{cases} \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2}\right)^2 & \textit{nếu n chẵn} \\ \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{\lfloor n/2 \rfloor}\right)^2 \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} & \textit{nếu n lẻ} \end{cases}$$

Giải thuật

<pre>int fib(int n) {
   F[2][2] = {{1, 1},{1, 0}};
   if (n == 0) return 0;
   power(F, n - 1);
   return F[0][0];
}
 	
void power(int F[2][2], int n) {
   if (n ≤ 1)
      return;
   T[2][2] = {{1, 1},{1, 0}};
   power(F, n / 2);
   multiply(F, F);
   if (n % 2 != 0)
      multiply(F,T);
}</pre>
	<pre>void multiply(F[2][2], T[2][2]) {
	  t1 = F[0][0]*T[0][0] + F[0][1]*T[1][0];
	  t2 = F[0][0]*T[0][1] + F[0][1]*T[1][1];
	  t3 = F[1][0]*T[0][0] + F[1][1]*T[1][0];
	  t4 = F[1][0]*T[0][1] + F[1][1]*T[1][1];
	  F[0][0] = t1;
	  F[0][1] = t2;
	  F[1][0] = t3;
	  F[1][1] = t4;
	}
	
	void main() {
	   cout << fib(5);
	}</pre>

Nhược điểm của cách cài đặt trên là chi phí quản lý ngăn xếp thời gian thực thi sẽ cao. Để cải tiến, chương trình dưới đây sử dụng vòng lặp:

```
int  Fibonacci(int n) {
   i = 1;    j = 0;
   k = 0;    h = 1;
   while (n) {
      if (n % 2) {
         t = j * h;
         j = i * h + j * k + t;
         i = i * k + t;
      }
      t = h * h;
      h = 2 * k * h + t;
      k = k * k + t;
      n = n / 2;
   }
   return j;
}
```

Chương 3: Kỹ thuật “Brute Force”

Giới thiệu chung

Chủ đề của chương này là nhóm giải thuật *brute force*, được xem là chiến lược thiết kế giải thuật đơn giản nhất.

Ví dụ: Tính x^n. Đơn giản là chỉ việc thực hiện dãy n phép nhân: $1 \times x \times ... \times x$.

Vai trò của brute force

1. Khả năng ứng dụng trong rất nhiều loại vấn đề, so với một vài chiến lược khác.
2. Chọn *brute force* nếu kích thước dữ liệu là nhỏ.
3. Nếu với yêu cầu xử lý một vài thể hiện của dữ liệu nhập và giữa một bên là chi phí tốn kém để xây dựng giải thuật thông minh, hiệu quả với một bên là *brute force* rẻ tiền, thời gian thực thi chấp nhận được thì lựa chọn sẽ là *brute force*.
4. Là tiêu chuẩn cơ sở (*bottom line*) để giúp đánh giá các giải thuật khác xử lý cùng vấn đề.
5. Tiền đề cho các cải tiến sau này.

Ví dụ: Sắp xếp nổi bọt

```
Bubblesort(a[1 .. n]) {
   for (i = 2; i ≤ n; i++)
      for (j = n; j ≥ i; j--)
         if (a[j - 1] > a[j])
            a[j - 1] ⇆ a[j];
}
```

Ví dụ: Tìm kiếm chuỗi con

```
SequentialStringSearch(T[1 .. n], P[1 .. m]) {
   count = 0;
   for (i = 1; i ≤ n - m + 1; i++) {
      j = 0;
      while (j < m) && (P[1 + j] == T[i + j])
         j++;
      if (j == m)
         count++;
   }
   return count;
}
```

Bài toán Tìm tổng (của) dãy con liên tục (có giá trị) lớn nhất

Phát biểu: Cho dãy n số nguyên: $a_1, a_2, \ldots, a_n$. Hãy tìm (và nhận diện dãy con tương ứng) giá trị lớn nhất của $\sum_{k=i}^{j} a_k$ với $1 \leq i \leq k \leq j \leq n$. Nếu mọi số nguyên của dãy đều là số âm thì trả về 0.

Giải thuật

```
MaxContSubSum(a[1 .. n]) {
   maxSum = 0;
   for (i = 1; i ≤ n; i++)
      for (j = i; j ≤ n; j++) {
         curSum = 0;
         for (k = i; k ≤ j; k++)
            curSum += a[k];
         if (curSum > maxSum)
            maxSum = curSum;
      }
   return maxSum;
}
```

Đánh giá:

$$A(n) \in \Theta(n^3)$$

Giải thuật (cải tiến 1)

```
MaxContSubSum(a[1 .. n]) {
   maxSum = 0;
   for (i = 1; i ≤ n; i++) {
      curSum = 0;
      for (j = i; j ≤ n; j++) {
         curSum += a[j];
         if (curSum > maxSum)
            maxSum = curSum;
      }
   }
   return maxSum;
}
```

Đánh giá:

$$A(n) \in \Theta(n^2)$$

Giải thuật (cải tiến 2)

```
MaxContSubSum(a[1 .. n]) {
   maxSum = curSum = 0;
   for (j = 1; j ≤ n; j++) {
      curSum += a[j];
      if (curSum > maxSum)
         maxSum = curSum;
      else
         if (curSum < 0)
            curSum = 0;
   }
   return maxSum;
}
```

Đánh giá:

$$A(n) \in \Theta(n)$$

Bài toán đổi tiền xu

Phát biểu: Giả sử có k mệnh giá tiền xu là $x_1, x_2, \ldots, x_k$. Tìm số lượng đồng tiền xu nhỏ nhất để có thể đổi n xu.
Chú ý: Luôn giả định rằng mệnh giá tiền xu nhỏ nhất là 1 xu.

Ví dụ: Các mệnh giá đồng tiền xu là 1 xu, 5 xu, 10 xu và 25 xu. Để đổi 72 xu, số đồng tiền cần ít nhất là: hai đồng 25 xu, hai đồng 10 xu và hai đồng 1 xu ($2 \times 25 + 2 \times 10 + 2 \times 1 = 72$).

Ý tưởng: Tìm tất cả các bộ $\langle c_1, c_2, \ldots, c_k \rangle$ sao cho:

$$c_1 \times x_1 + c_2 \times x_2 + \cdots + c_k \times x_k = n \ \wedge \ \sum_{i=1}^{k} c_i \, \textit{nhỏ nhất}$$

với c_i là số lượng đồng xu có mệnh giá x_i.

Đánh giá:

$$T(n) = \frac{n}{x_1} \times \frac{n}{x_2} \times \ldots \times \frac{n}{x_k} = \frac{n^k}{x_1 \times x_2 \times \ldots \times x_k} \in \Theta(n^k)$$

khi n đủ lớn vì $x_1, x_2, \ldots, x_k$ là các hằng số.

Bài toán Cặp điểm gần nhất

Phát biểu: Tìm cặp điểm gần nhất trên mặt phẳng tọa độ hình chữ nhật (*Cartesian coordinate system*) theo độ đo Euclid.

Giải thuật

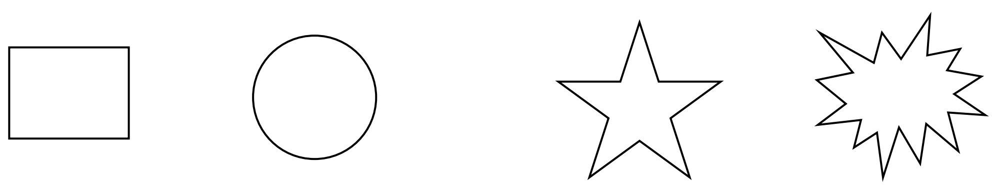
```
BruteForceClosestPoints(P[1 .. n]) {
   dmin = ∞;
   for (i = 1; i ≤ n - 1; i++)
      for (j = i + 1; j ≤ n; j++) {
         d = \sqrt{(P[i].x - P[j].x)^2 + (P[i].y - P[j].y)^2};
         if (d < dmin) {
            dmin = d;
            point1 = P[i];
            point2 = P[j];
         }
      }
   return   <point1, point2>;
}
```

Đánh giá: $\Theta(n^2)$

Bài toán Đa giác lồi

Phát biểu: Cho tập điểm S, tìm đa giác lồi chứa tất cả các điểm thuộc về tập này.

Ví dụ: Hai hình bên trái là đa giác lồi, hai hình bên phải thì không.



Nhận xét:

"Đoạn thẳng $\overline{pq}$ nối hai điểm p và q của tập điểm S là cạnh của đa giác lồi nếu và chỉ nếu tất cả các điểm khác của S cùng nằm về một phía của mặt phẳng được phân cách bởi $\overline{pq}$".

Biết rằng đường thẳng đi qua hai điểm (x_1, y_1) và (x_2, y_2) được xác định bằng công thức:

$$ax + by = c$$

với $a = y_2 - y_1, b = x_1 - x_2, c = x_1y_2 - x_2y_1$.

Giải thuật

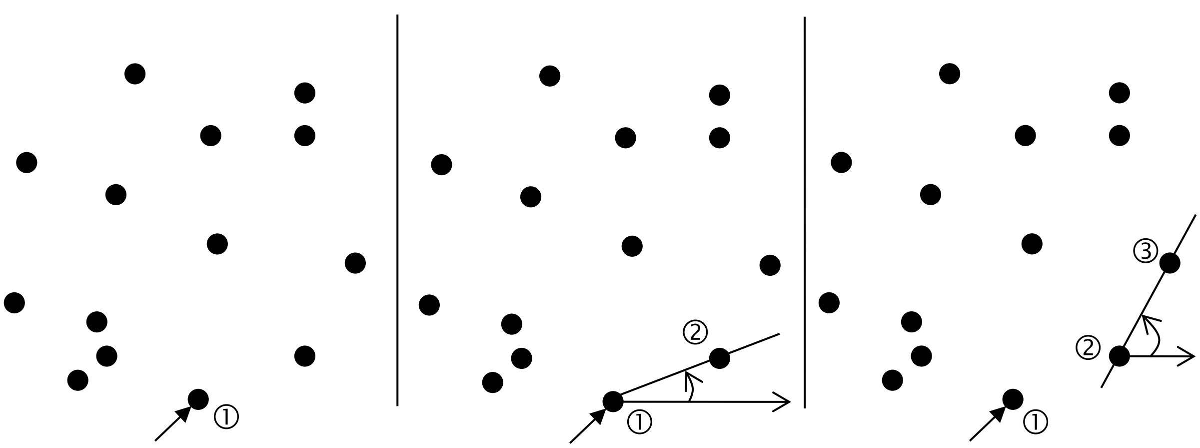
```
for (Mỗi điểm p_i ∈ S: i = 1 → n - 1)
   for (Mỗi điểm q_j ∈ S: j = i + 1 → n) {
      Xây dựng đường thẳng p_iq_j;
      if (Tất cả các điểm khác trong S nằm về một phía của đường p_iq_j)
         Bổ sung cặp điểm <p_i, q_j> vào danh sách kết quả;
   }
```

Đánh giá: $\Theta(n^3)$

Mở rộng: Cũng với tiếp cận *brute force*, nhưng lần này, đầu ra của giải thuật là danh sách điểm cực theo thứ tự ngược chiều kim đồng hồ.

Giả sử p là điểm cực vừa được phát hiện và *curAngle* là góc hiện tại. Điểm cực q kế tiếp sẽ là điểm thỏa điều kiện:

$$curAngle \leq (\widehat{\overline{pq}, 0x}) < (\widehat{\overline{pr}, 0x}), \forall r \in S: r \neq p, r \neq q$$



Giải thuật

```
computeAngle(point from, point to) {
   angle = atan2(to.y - from.y, to.x - from.x);
   if (angle < 0)
      angle += 2 * π;
   return angle;
}

findNextExtremePoint(S, cur, curAngle) {
   minAngle = 2 * π;
   S \= cur;
   for (Mỗi p trong S) {
      angle = computeAngle(cur, p);
      if (angle < minAngle && angle ≥ curAngle) {
         next = p;
         minAngle = angle;
      }
   }
   S ∪= cur;
   return [next, minAngle];
}

computeConvexHull(S) {
   convexHull = ∅;
   // Gọi first là điểm có tung độ nhỏ nhất trong S;
   convexHull ∪= first;
   curAngle = 0;
   point cur = first;
   while (true) {
      [next, curAngle] = findNextExtremePoint(S, cur, curAngle);
      if (first == next)
         break;
      convexHull ∪= next;
      cur = next;
   }
   return convexHull;
}
```

Đánh giá: Trong trường hợp xấu nhất thì chi phí là $\Theta(n^2)$. Nếu số lượng điểm cực nhỏ hơn $\log n$ có chi phí $\Theta(n \log n)$.

Tìm kiếm vét cạn

Tìm kiếm vét cạn đơn giản là một dạng của tiếp cận brute force đối với các vấn đề tổ hợp.

Giải thuật tạo tập hợp của các tập con từ tập có kích thước n

```
for (k = 0; k < 2ⁿ; k++)
   In chuỗi bit chiều dài n biểu diễn k;
```

Giải thuật tạo hoán vị

```
Taohoanvi(pivot, a[1 .. n]) {
   if (pivot == n)
      inhoanvi(a);
   else
      for (i = pivot; i ≤ n; i++) {
         a[pivot] ⇆ a[i];
         Taohoanvi(pivot + 1, a);
         a[pivot] ⇆ a[i];
      }
}
Taohoanvi(1, a);
```

Ví dụ: Xét các hoán vị của dãy bốn số a_1, a_2, a_3, a_4

a_1, a_2, a_3, a_4	a_2, a_1, a_3, a_4	a_3, a_2, a_1, a_4	a_4, a_2, a_3, a_1
a_1, a_2, a_4, a_3	a_2, a_1, a_4, a_3	a_3, a_2, a_4, a_1	a_4, a_2, a_1, a_3
a_1, a_3, a_2, a_4	a_2, a_3, a_1, a_4	a_3, a_1, a_2, a_4	a_4, a_3, a_2, a_1
a_1, a_3, a_4, a_2	a_2, a_3, a_4, a_1	a_3, a_1, a_4, a_2	a_4, a_3, a_1, a_2
a_1, a_4, a_3, a_2	a_2, a_4, a_3, a_1	a_3, a_4, a_1, a_2	a_4, a_1, a_3, a_2
a_1, a_4, a_2, a_3	a_2, a_4, a_1, a_3	a_3, a_4, a_2, a_1	a_4, a_1, a_2, a_3

Đánh giá: Gọi $T(n)$ là chi phí để tạo ra mọi hoán vị của *n* phần tử. Hệ thức truy hồi là:

$$T(n) = nT(n-1) + \Theta(n) \text{ và } T(1) = 0$$

với $\Theta(n)$ là chi phí cho việc hoán vị cặp giá trị a_{pivot} và a_i trong vòng `for`.

$$T(n) \in \Omega(n!)$$

Bài toán đường đi người bán hàng

Phát biểu: Cho n thành phố. Tìm con đường *ngắn nhất* trong số các con đường đi qua mọi thành phố (duy nhất một lần) và quay trở về thành phố xuất phát.

Chu trình Hamilton: Cho đồ thị có n đỉnh. Một chu trình Hamilton là dãy của $n + 1$ đỉnh kề nhau: $v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}, v_{i_0}$, với đỉnh đầu và đỉnh cuối của dãy là một, các đỉnh còn lại khác nhau từng đôi một.

Ý tưởng:

– Chọn đỉnh xuất phát v_{i_0}. Xây dựng tất cả các hoán vị của $n - 1$ đỉnh còn lại (trung gian): $v_{i_1}, \ldots, v_{i_{n-1}}$.

– Với mỗi hoán vị, bổ sung đỉnh v_{i_0} vào đầu và cuối rồi kiểm tra xem từng cặp đỉnh liên tiếp có phải là đỉnh kề?

– Nếu tất cả các cặp đỉnh trong hoán vị đều là đỉnh kề thì đây là một chu trình Hamilton. Có thể tính chiều dài để xử lý.

– Ngược lại thì chuyển sang hoán vị kế.

Đánh giá: $\Theta(n!)$

Tổng các tập con

Phát biểu: Tìm tập con của tập đã cho $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ gồm n số *nguyên dương* sao cho tổng các phần tử của tập con này bằng với k (nguyên dương).

Tiếp cận *Brute-force* đơn giản sẽ tìm mọi tập con có thể và tính tổng của mỗi tập con.

Đánh giá: $\Theta(2^n)$.

Bài toán túi xách 0-1

Phát biểu: Một chiếc túi có khả năng chứa khối lượng tối đa W. Có n vật với khối lượng $w_1, w_2, \ldots, w_n$ và giá trị tương ứng là $v_1, v_2, \ldots, v_n$. Tìm tập con có giá trị nhất mà chiếc túi có khả năng mang được.

$$maximize \sum_{i=1}^{n} v_i x_i$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \le W\ where\ x_i = \{0, 1\}, i = 1, .., n$$

Tiếp cận vét cạn cho vấn đề này yêu cầu phát sinh mọi tập con có thể của tập n vật đã cho. Khi đó, tính toán khối lượng của mỗi tập con. Khối lượng nào lớn hơn W thì bỏ qua. Trong số những khối lượng còn lại, chọn cái có giá trị cao nhất.

Đánh giá: $\Theta(2^n)$.

Bài toán kết nối

Phát biểu: Có n người và n việc đang tìm nhau. Biết rằng, người thứ i khi có việc thứ j sẽ nhận mức lương $C(i, j)$. Vấn đề là tìm cách giao việc thế nào để tổng chi phí là thấp nhất.

Ví dụ: Bảng C dưới đây chỉ ra các thông tin liên quan

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

Kết quả của bài toán có thể biểu diễn như một dãy $J = \langle j_1, j_2, \ldots, j_n \rangle: \forall i \in [1, n], j_i \in [1, n]$ với ngữ nghĩa là người thứ *i* trong dãy J nhận việc j_i, có mức lương $C(i, j_i)$.

Ví dụ: <2, 4, 1, 3> nghĩa là người 1 nhận việc 2, người 2 nhận việc 4, người 3 nhận việc 1 và người 4 nhận việc 3.

Ý tưởng: Phát sinh mọi hoán vị có thể của dãy *n* số từ 1 đến *n*. Xem như đây là một dãy *J* rồi tính tổng chi phí và xác định hoán vị nào có chi phí thấp nhất.

Đánh giá: $\Theta(n!)$

Chương 4: Kỹ thuật Quay lui và Nhánh cận

Phần 1: Kỹ thuật quay lui

Giới thiệu chung

Giải thuật quay lui (*backtracking*) thường được sử dụng cho các bài toán tổ hợp. Với loại bài toán này thì phạm vi của những *lời giải tiềm năng* là vô cùng lớn ($\in \Omega(2^n)$). Trong số đó, sẽ có một tập các *lời giải khả thi*, vốn được cấu thành từ nhiều *lời giải thành phần*.

Quay lui được xem như là một cải tiến của tìm kiếm vét cạn (*exhaustive search*).

Cây không gian các trạng thái

Giải thuật quay lui *thường* dựa trên việc xây dựng (không tường minh) cái gọi là Cây không gian các trạng thái (*state-space tree*). Mỗi một nút của cây thể hiện giá trị của một *lời giải thành phần* của (các) *lời giải tiềm năng*. Gốc của cây biểu diễn trạng thái bắt đầu trước khi tiến hành xử lý. Thông thường, *một con đường đi từ gốc đến lá của cây là một lời giải tiềm năng*.

Dạng thức thứ nhất của giải thuật Quay lui:

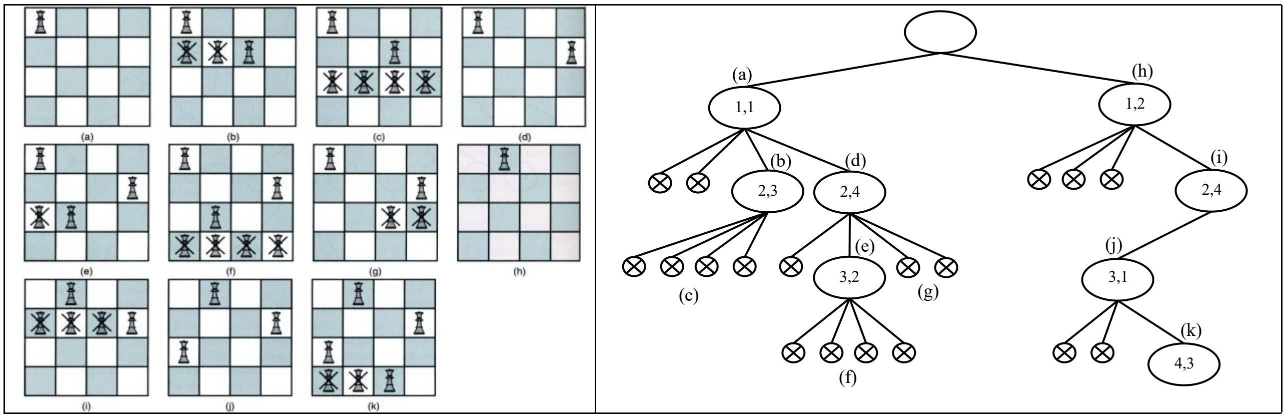
```
Backtracking(u) {
   if (promising(u))
      if (u là nút lá)
         Xuất lời giải;
      else
         for (Mỗi nút con v của u)
            Backtracking(v);
}
```

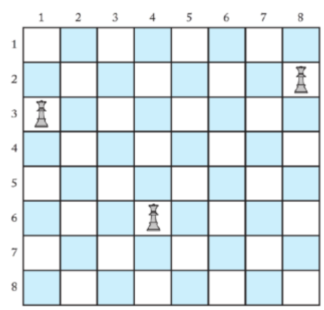
Dạng thức thứ hai của giải thuật Quay lui:

```
Backtracking(u) {
   for (Mỗi nút con v của u)
      if (promising(v))
         if (v là nút lá)
            Xuất lời giải;
         else
            Backtracking(v);
}
```

Bài toán n–Hậu

Phát biểu: Đặt *n* quân Hậu lên bàn cờ *n*×*n* để chúng không thể tấn công lẫn nhau.





Giải thuật

```
promising(i) {
   j = 1;   flag = true;
   while (j < i && flag) {
      if (col[i] == col[j] || abs(col[i] - col[j]) == i - j)
         flag = false;
      j++;
   }
   return   flag;
}
Version 1
n_Queens(i) {
   if (promising(i))
      if (i == n)
         print(col[1 .. n]);
      else
         for (j = 1; j ≤ n; j++) {
            col[i + 1] = j;
            n_Queens(i + 1);
         }
}
n_Queens(0);
```

```
Version 2
n_Queens(i) {
   for (j = 1; j ≤ n; j++) {
      col[i] = j;
      if (promising(i))
         if (i == n)
            print(col[1 .. n]);
         else
            n_Queens(i + 1);
   }
}
n_Queens(1);
```

Bài toán Ngựa đi tuần

Phát biểu: Đặt quân ngựa lên một ô bất kỳ $\langle r_0, c_0\rangle$ của bàn cờ $n \times n$. Hãy chỉ ra mọi hành trình (nếu có), đi qua tất cả các ô đúng một lần. Biết rằng quân ngựa có khả năng di chuyển tối đa đến 8 vị trí xung quanh nó.

↙	4	←	3	↖	-2
5				2	-1
↓		♞		↑	0
6				1	1
↘	7		0	↗	2
-2	-1	0	1	2	

Giải thuật

```
KnightTour(i, r, c) {
   for (k = 1; k ≤ 8; k++) {
      u = r + row[k];
      v = c + col[k];
      if ((1 ≤ u, v ≤ n) && (h[u][v] == 0)) {
         h[u][v] = i;
         if (i == n^2)
            print(h);
         else
            KnightTour(i + 1, u, v);
         h[u][v] = 0;
      }
   }
}
h[r0][c0] = 1;
KnightTour(2, r0, c0);
```

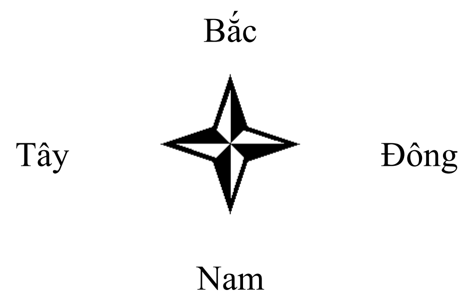
Bài toán Tìm đường trong mê cung

Phát biểu: Một người máy (robot) được yêu cầu tìm đường đi trong mê cung. Nó được đặt ở vị trí ban đầu nào đó trong mê cung và được lệnh tìm một vị trí khác cũng trong mê cung này (gọi là *đích* hay *cổng thoát*).

Tại mỗi thời điểm, người máy chỉ có thể đi một bước theo một trong bốn hướng (duyệt lần lượt): Bắc, Đông, Nam, Tây và không cho phép vượt ra ngoài phạm vi của mê cung.

#	S	#	#	.	#
#	.	.	.	.	#
.	#	.	#	.	#
.	.	.	.	#	#
.	.	#	#	.	G
#	.	.	.	.	#
#	#	#	.	#	#

(a)



#	S	#	#	.	#
#	×	×	×	×	#
.	#	.	#	.	#
.	.	.	.	#	#
.	.	#	#	.	G
#	.	.	.	.	#
#	#	#	.	#	#

(b)

#	S	#	#	.	#
#	×	×	.	.	#
.	#	×	#	.	#
.	×	×	.	#	#
.	×	#	#	×	G
#	×	×	×	×	#
#	#	#	.	#	#

(c)

1	2	3	4	5	6
××#### #×#..# #×#..# #××#.# ###... G...##	××#### #×#..# #×#..# #××#.# ###... G...##	××#### #×#..# #×#..# #××#.# ###... G...##	××#### #×#..# #×#..# #××#.# ###... G...##	××#### #×#..# #×#..# #××#.# ###... G...##	××#### #×#..# #×#..# #×.#.# ###... G...##

Giải thuật

```
bool  Find_Path(r, c) {
   if ((r, c) ∉ Maze)          return      false;
   if (Maze[r][c] == 'G')      return      true;
   if (Maze[r][c] == 'x')      return      false;
   if (Maze[r][c] == '#')      return      false;
   Maze[r][c] = 'x';
   if (Find_Path(r - 1, c) == true)        return      true;
   if (Find_Path(r, c + 1) == true)        return      true;
   if (Find_Path(r + 1, c) == true)        return      true;
   if (Find_Path(r, c - 1) == true)        return      true;
   Maze[r][c] = '.';
   return   false;
}
Find_Path(r0, c0);
```

Chu trình Hamilton

Giải thuật

```
bool promising(int pos, int v) {
   if (pos == n && G[v][path[1]] == 0)
      return false;
   else
      if (G[path[pos - 1]][v] == 0)
         return false;
      else
         for (int i = 1; i < pos; i++)
            if (path[i] == v)
               return false;
   return true;
}
Hamiltonian(bool G[1..n][1..n], int path[1..n], int pos) {
   if (pos == n + 1)
      print(path);
   else
      for (v = 1; v ≤ n; v++)
         if (promising(pos, v)) {
            path[pos] = v;
            Hamiltonian(G, path, pos + 1);
         }
}
path[1 .. n] = -1;
path[1] = 1;
Hamiltonian(G, path, 2);
```

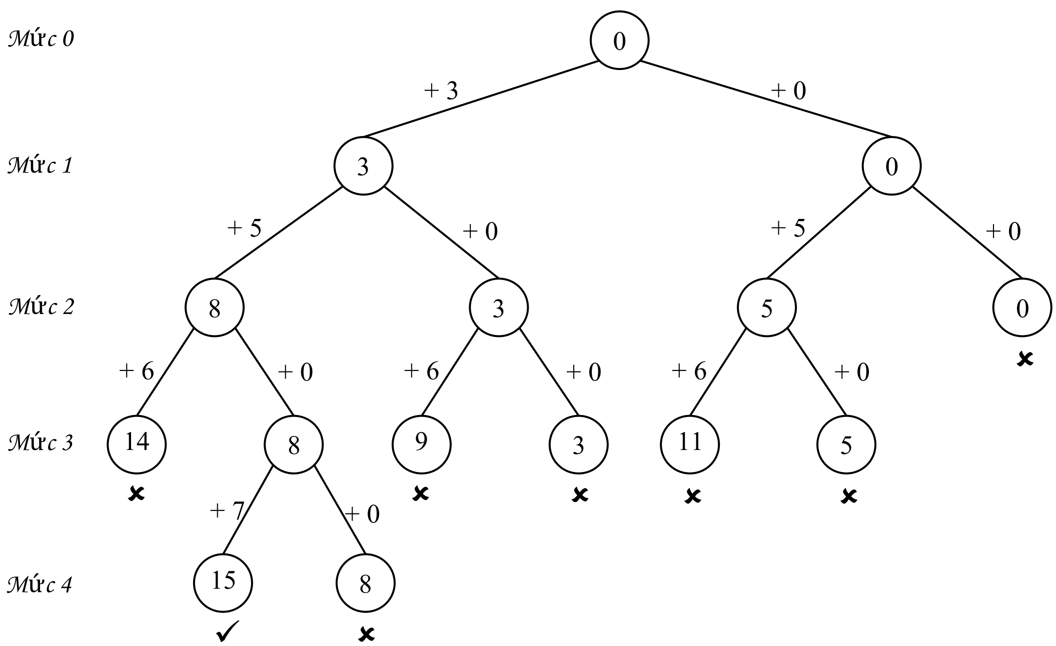
Bài toán Tổng các tập con

Phát biểu: Tìm tập con của tập đã cho $\mathcal{W} = \{w_1, w_2, \dots, w_n\}$ gồm n số nguyên dương (có thể bằng nhau) sao cho tổng các phần tử của tập con này bằng với t.

Dạng 1

Lời giải S sẽ là một vector kích thước n: $\{s_1, s_2, \dots, s_n\}$ với $s_i \in \{0,1\}$, tương ứng w_i có thuộc về tập con cần tìm hay không.

Ví dụ: $\mathcal{W} = \{3, 5, 6, 7\}$ và $t = 15$.



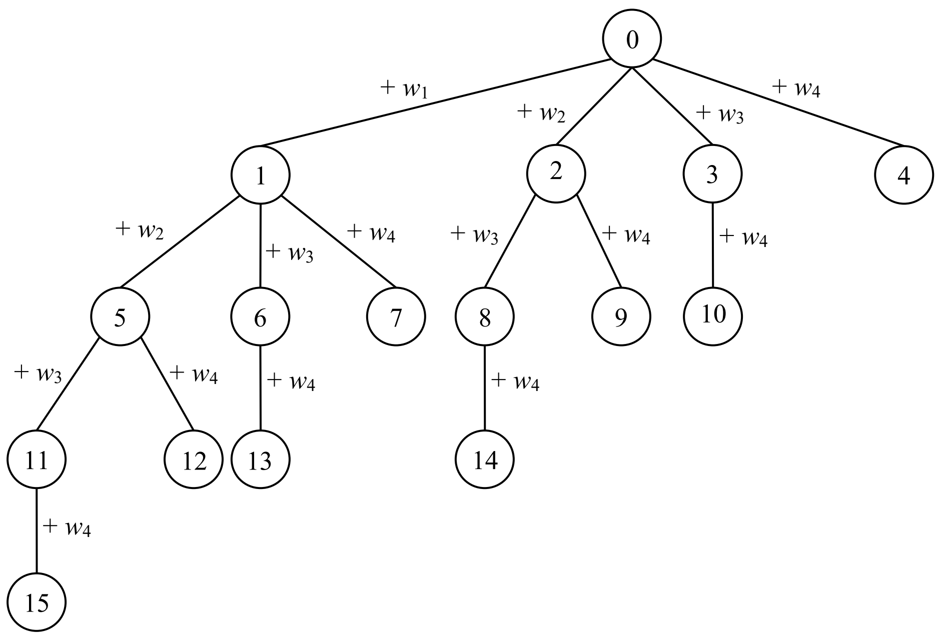
Giải thuật

```
SoS(k, sum, total, w[1 .. n], s[1 .. n]) {
   if (sum == t)
      print(s);
   else
      if ((sum + total ≥ t) && (sum + w[k] ≤ t)) {
         s[k] = true;
         SoS(k + 1, sum + w[k], total - w[k], w, s);
         s[k] = false;
         SoS(k + 1, sum, total - w[k], w, s);
      }
}
bool  s[1 .. n] = {false};
total = ∑_{i=1}^{n} w[i];
sort(w);
if (w[1] ≤ t ≤ total)
   SoS(1, 0, total, w, s);
```

Dạng 2

Một lời giải S sẽ là tập hợp của các vật được chọn.

Giả sử tập ban đầu có 4 giá trị: $\mathcal{W} = \{w_1, w_2, w_3, w_4\}$. Cây không gian các trạng thái được xây dựng như sau:



Giải thuật (Quay lui vét cạn)

```
SoS(s[1 .. n], size, sum, start) {
   if (sum == t)
      print(s, size);
   else
      for (i = start; i ≤ n; i++) {
         s[size] = w[i];
         SoS(s, size + 1, sum + w[i], i + 1);
      }
}
s[1 .. n] = {0};
total = ∑_{i=1}^{n} w[i];
if (w[1] ≤ t ≤ total)
   SoS(s, 1, 0, 1);
```

Giải thuật (cải tiến)

```
SoS(s[1 .. n], size, sum, start, total) {
   if (sum == t)
      print(s, size);
   else {
      lost = 0;
      for (i = start; i ≤ n; i++) {
         if ((sum + total - lost ≥ t) && (sum + w[i] ≤ t)) {
            s[size] = w[i];
            SoS(s, size + 1, sum + w[i], i + 1, total - lost - w[i]);
         }
         lost += w[i];
      }
   }
}
s[1 .. n] = {0};
total = ∑_{i=1}^{n} w[i];
sort(w);
if (w[1] ≤ t ≤ total)
   SoS(s, 1, 0, 1, total);
```

Chương 5: Kỹ thuật Chia để trị (Divide-and-Conquer)

Giới thiệu chung

Chia để trị là kỹ thuật thiết kế giải thuật rất thông dụng, được thể hiện qua 3 bước chung sau:

Bước 1. Thực thể ban đầu của vấn đề được chia thành nhiều thực thể nhỏ hơn nhưng vẫn duy trì bản chất của vấn đề. Lý tưởng nhất là những thực thể nhỏ hơn có kích thước bằng nhau.

Bước 2. Tiếp tục giải quyết những thực thể nhỏ hơn theo cách của Bước 1 (đệ qui) cho đến khi thực thể đủ nhỏ thì xử lý thật sự (có thể vận dụng một giải thuật khác trong trường hợp này).

Bước 3. Lời giải của những thực thể nhỏ hơn sẽ được tổ hợp lại dần để cuối cùng cho ra lời giải của thực thể ban đầu của vấn đề.

Ví dụ: Tìm số lớn nhất của dãy n số (để đơn giản, xem $n = 2^k$).

Hệ thức truy hồi chia để trị

Gọi n là kích thước nguyên thủy của thực thể ban đầu của bài toán. Chia bài toán thành $a(> 1)$ thực thể, mỗi cái có kích thước ${}^{n}/_{b}$ (lý tưởng là ${}^{n}/_{a}$) với $b > 1$. *Hệ thức truy hồi chia để trị*, biểu diễn thời gian thực thi $T(n)$ là:

$$T(n) = aT\left({}^{n}/_{b}\right) + f(n)$$

với $f(n)$ là hàm tính thời gian dành cho việc phân chia và tổng hợp kết quả.

<u>Định lý chủ</u> (*Master theorem*) Cho hệ thức truy hồi chia để trị $T(n) = aT\left({}^{n}/_{b}\right) + f(n)$. Nếu $f(n) \in \Theta(n^d), d \geq 0$ thì:

$$T(n) \in \begin{cases} \Theta(n^d) & a < b^d \\ \Theta(n^d \log n) & a = b^d \\ \Theta\left(n^{\log_b a}\right) & a > b^d \end{cases}$$

Kết quả thu được tương tự đối với O và Ω.

Ví dụ: Tìm số lớn nhất trong dãy n số (để đơn giản, xem $n = 2^k$).

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

Ví dụ: Tìm đồng thời số nhỏ nhất và lớn nhất của mảng một chiều có n phần tử.

Giải thuật

<pre>MinMax(a[1..n]) {
   min = max = a[1];
   for (i = 2; i ≤ n; i++) {
      if (a[i] > max)
         max = a[i];
      if (a[i] < min)
         min = a[i];
   }
   return   <min, max>
}</pre>
	<pre>MinMax(a[1..n]) {
	   min = max = a[1];
	   for (i = 2; i ≤ n; i++)
	      if (a[i] > max)
	         max = a[i];
	      else
	         if (a[i] < min)
	            min = a[i];
	   return <min, max>
	}</pre>

Giải thuật

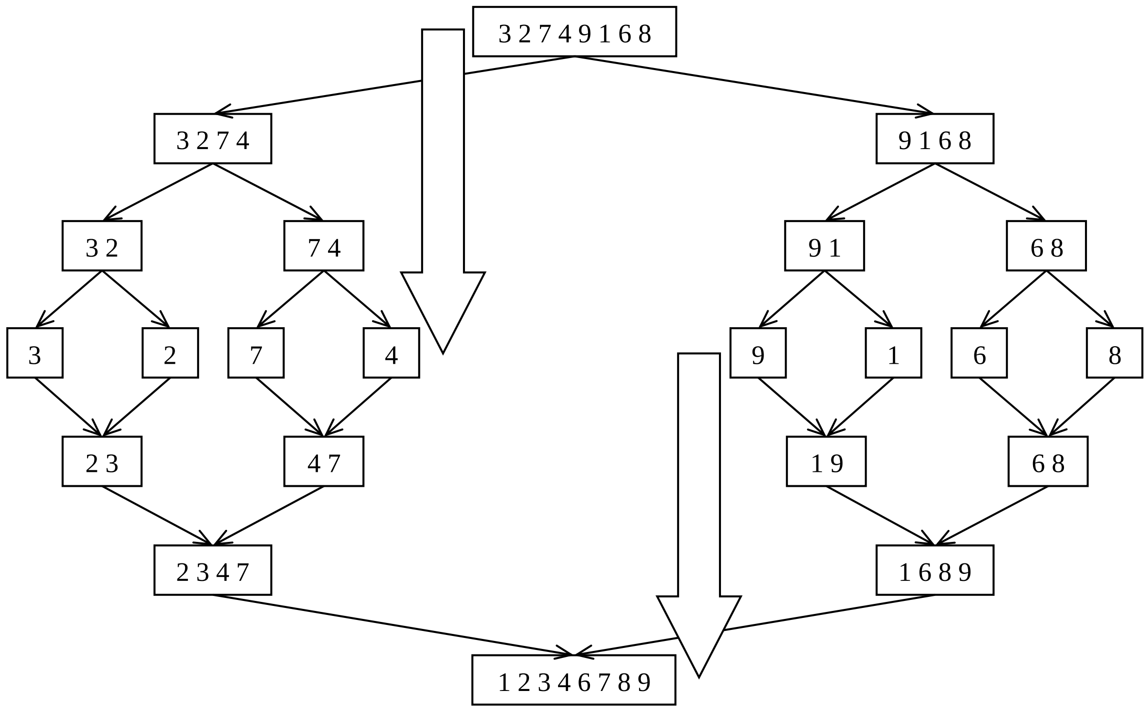
```
MinMaxDC(l, r, & min, & max) {
   if (l ≥ r - 1) // (l == r) || (l == r - 1)
      if (a[l] < a[r])   {
         min = a[l];
         max = a[r];
      }
      else {
         min = a[r];
         max = a[l];
      }
   else {
      m = ⌊(l + r) / 2⌋;
      MinMaxDC(l, m, minL, maxL);
      MinMaxDC(m + 1, r, minR, maxR);

      min = (minL < minR) ? minL : minR;
      max = (maxL < maxR) ? maxL : maxR;
   }
}
```

Đánh giá: Hệ thức truy hồi chia để trị là:

$$C(n) = \begin{cases} C\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + C\left(n - \left\lfloor\frac{n}{2}\right\rfloor\right) + 2 & n > 2 \\ 1 & n \le 2 \end{cases}$$

Sắp xếp trộn (Mergesort)

Ý tưởng: Gồm hai giai đoạn: tách (chia) và trộn (tổ hợp kết quả thành phần).




```
mergeSort(a[1 .. n], low, high) {
   if (low < high) {
      mid = ⌊(low + high) / 2⌋;
      mergeSort(a, low, mid);
      mergeSort(a, mid + 1, high);
      merge(a, low, mid, high);
   }
}
merge(a[1 .. n], low, mid, high) {
   i = low;
   j = mid + 1;
   k = low;
   while (i ≤ mid) && (j ≤ high)
      if (a[i] ≤ a[j])
         buf[k ++] = a[i ++];
      else
         buf[k ++] = a[j ++];
   if (i > mid)
      buf[k .. high] = a[j .. high];
   else
      buf[k .. high] = a[i .. mid];
   a[low .. high] = buf[low .. high];
}
mergeSort(a, 1, n);
```

Đánh giá

- Nếu chọn phép so sánh là thao tác cơ sở:

  Trường hợp tốt nhất: Khi đó, hệ thức truy hồi là:

$$T(n) = \begin{cases} T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \left\lfloor \frac{n}{2} \right\rfloor & n > 1 \\ 0 & n = 1 \end{cases}$$

  Trường hợp xấu nhất: Hệ thức truy hồi là:

$$T(n) = \begin{cases} T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + (n-1) & n > 1 \\ 0 & n = 1 \end{cases}$$

- Nếu thao tác cơ sở là phép chuyển dời:
  Hệ thức truy hồi (không phân chia trường hợp tốt hay xấu nhất) luôn là:

$$M(n) = \begin{cases} M\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + M\left(\left\lceil \frac{n}{2} \right\rceil\right) + n & n > 1 \\ 0 & n = 1 \end{cases}$$

Sắp xếp Nhanh (Quicksort)

Dãy $a_1, a_2, \ldots, a_n$ được gọi là đã phân hoạch sau một lượt duyệt nếu tại một vị trí s nào đó, các phần tử đứng trước có giá trị nhỏ hơn hoặc bằng a_s, các phần tử đứng sau có giá trị lớn hơn hoặc bằng a_s:

$$\{a_1 \ldots a_{s-1}\} \leq a_s \leq \{a_{s+1} \ldots a_n\}$$

Như vậy, a_s đã được đặt đúng vị trí của nó và phép phân hoạch lại được thực hiện trên hai dãy con: $a_1, \ldots, a_{s-1}$ và $a_{s+1} \ldots a_n$. Điều này được tiếp diễn tương tự (đệ qui) cho đến khi dãy cần phân hoạch chỉ chứa *một* phần tử.

Giải thuật

```
Quicksort(a[left .. right]) {
   if (left < right){
      s = Partition(a[left .. right]);
      Quicksort(a[left .. s - 1]);
      Quicksort(a[s + 1 .. right]);
   }
}
```

Giải thuật

```
Partition(a[left .. right]) {
   p = a[left];
   i = left;
   j = right + 1;
   do {
      do i++; while (a[i] < p);
      do j--; while (a[j] > p);
      swap(a[i], a[j]);
   } while (i < j);

   swap(a[i], a[j]);
   swap(a[left], a[j]);

   return   j;
}
```

Câu hỏi: Liệu thiết kế trên có an toàn không?

Đánh giá

Để đơn giản, xem như các phần tử trong dãy là khác nhau từng đôi một và kích thước dãy dữ liệu là $n = 2^k$. Thao tác cơ sở là phép so sánh trong vòng lặp `do..while`.

Trường hợp tốt nhất:

$$C_b(n) = \begin{cases} 2C_b\left(\frac{n}{2}\right) + \Theta(n) & n > 1 \\ 0 & n = 1 \end{cases}$$

$$C_b(n) \in \Theta(n \log n)$$

Trường hợp xấu nhất:

$$C_w(n) \in \Theta(n^2)$$

Trường hợp trung bình:

$$C(n) \approx 1.39n \log_2 n$$

Nhân số lớn

Ý tưởng: Gauss nhận thấy, khi nhân hai số phức

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

và biết rằng:

$$bc + ad = (a + b)(c + d) - (ac + bd)$$

Ví dụ: Xét phép nhân của hai số 12 × 34. Hai số này có thể biểu diễn là:

$12 = 1 \times 10^1 + 2 \times 10^0$ và $34 = 3 \times 10^1 + 4 \times 10^0$

$12 \times 34 = (1 \times 10^1 + 2 \times 10^0) \times (3 \times 10^1 + 4 \times 10^0)$

$= (1 \times 3) \times 10^2 + (1 \times 4 + 2 \times 3) \times 10^1 + (2 \times 4) \times 10^0$

Dựa vào ý tưởng của Gauss, có thể thay thế:

$$1 \times 4 + 2 \times 3 = (1 + 2) \times (3 + 4) - (1 \times 3 + 2 \times 4) = 10$$

Phát biểu: Giả sử đã xây dựng kiểu dữ liệu `large_integer` với các phép tính: `mul 10`^{m}, `div 10`^{m}, `mod 10`^{m} với chi phí tuyến tính. Hãy thực hiện phép nhân hai số có n chữ số (với n bất kỳ và lớn hơn ngưỡng α): $u \times v$.

Giải thuật

```
large_integer MUL(large_integer u, v) {
   large_integer x, y, w, z;
   n = max(Số chữ số của u, Số chữ số của v);
   if (u == 0 || v == 0)
      return 0;
   else
      if (n ≤ α)
         return u * v;  // built-in operator
      else {
         m = ⌊n / 2⌋;
         x = u div 10^m;  y = u mod 10^m;
         w = v div 10^m;  z = v mod 10^m;
         return   MUL(x, w) mul 10^2m +
                 (MUL(x, z) + MUL(y, w)) mul 10^m +
                  MUL(y, z);
      }
}
```

Đánh giá: Hệ thức truy hồi chia để trị trong trường hợp này là:

$$T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + \Theta(n) & n > \alpha \\ 1 & n \le \alpha \end{cases}$$

Giải thuật (cải tiến)

```
large_integer MUL(large_integer u, v, n) {
   n = max(Số chữ số của u, Số chữ số của v);
   if (u == 0 || v == 0)
      return 0;
   else
      if (n ≤ α)
         return u × v;
      else {
         m = ⌊n / 2⌋;
         x = u div 10^m;  y = u mod 10^m;
         w = v div 10^m;  z = v mod 10^m;
         r = MUL(x + y, w + z);
         p = MUL(x, w);
         q = MUL(y, z);
         return p mul 10^2m + (r − p − q) mul 10^m + q;
      }
}
```

Đánh giá: Hệ thức truy hồi chia để trị trong trường hợp này là:

$$T(n) = \begin{cases} 3T\left(\frac{n}{2}\right) + \Theta(n) & n > \alpha \\ 1 & n \leq \alpha \end{cases}$$

Vì $d = 1, a = 3, b = 2$ nên $\Theta\left(n^{\log_2 3}\right) \approx \Theta(n^{1.585})$.

Nhân ma trận (của) Strassen

Ý tưởng xuất phát từ việc nhân hai ma trận kích thước 2×2 như sau:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$
$$= \begin{bmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} \times b_{11} + a_{22} \times b_{21} & a_{21} \times b_{12} + a_{22} \times b_{22} \end{bmatrix}$$

Như vậy, số lượng phép × phải thực hiện là 8 và phép + là 4.

Tuy nhiên, ma trận kết quả có thể tìm được như sau:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

với:

$m_1 = (a_{11} + a_{22}) \times (b_{11} + b_{22})$
$m_2 = (a_{21} + a_{22}) \times b_{11}$
$m_3 = a_{11} \times (b_{12} - b_{22})$
$m_4 = a_{22} \times (b_{21} - b_{11})$
$m_5 = (a_{11} + a_{12}) \times b_{22}$
$m_6 = (a_{21} - a_{11}) \times (b_{11} + b_{12})$
$m_7 = (a_{12} - a_{22}) \times (b_{21} + b_{22})$

Từ đây, số lượng phép × giảm xuống còn 7, phép +/– tăng lên là 18.

Gọi A và B là hai ma trận $n \times n$ (cho rằng $n = 2^k$). Chúng ta sẽ chia các ma trận này thành những ma trận kích thước $\frac{n}{2} \times \frac{n}{2}$ (kể cả ma trận kết quả *C*):

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Giải thuật

```
Strassen(n, A[1..n][1..n], B[1..n][1..n], C[1..n][1..n]) {
   if (n ≤ α)
      C = A × B;
   else {
      "Phân chia A thành A11, A12, A21, A22";
      "Phân chia B thành B11, B12, B21, B22";
      Strassen(n/2, A11 + A22, B11 + B22, M1);
      …
      Strassen(n/2, A12 – A22, B21 + B22, M7);
      C11 = M1 + M4 – M5 + M7;
      C12 = M3 + M5;
      C21 = M2 + M4;
      C22 = M1 + M3 – M2 + M6;
      Tổ_hợp(C, C11, C12, C21, C22);
   }
}
```

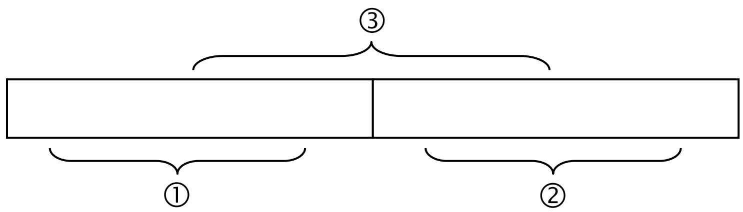
Đánh giá: Hệ thức truy hồi chia để trị là:

$$T(n) = \begin{cases} 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 & n > \alpha \\ 1 & n \leq \alpha \end{cases}$$

$$T(n) \in \Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2.81})$$

Bài toán Tìm tổng lớn nhất của dãy con liên tục

Phát biểu: Cho dãy số nguyên n phần tử: $a_1, a_2, \dots, a_n$. Hãy tìm (và nhận diện dãy con tương ứng) giá trị lớn nhất của $\sum_{k=i}^{j} a_k$ với $1 \le i \le k \le j \le n$.



Giải thuật

```
sumMax(a[1..n], l, r) {
   if (l == r)
      return   max(a[l], 0);

   c = ⌊(l + r) / 2⌋;
   maxLS = sumMax(a, l, c);
   maxRS = sumMax(a, c + 1, r);

   tmp = maxLpartS = 0;
   for (i = c; i ≥ l; i--) {
      tmp += a[i];
      if (tmp > maxLpartS)
         maxLpartS = tmp;
   }
   tmp = maxRpartS = 0;
   for (i = c + 1; i ≤ r; i++) {
      tmp += a[i];
      if (tmp > maxRpartS)
         maxRpartS = tmp;
   }
   tmp = maxLpartS + maxRpartS;
   return   max(tmp, maxLS, maxRS);
}
max = sumMax(a, 1, n);
```

Đánh giá:

Hệ thức truy hồi chia để trị trong trường hợp này là

$$T(n) = \begin{cases} T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & n > 1 \\ 0 & n = 1 \end{cases}$$

Bài toán cặp (điểm) gần nhất

Phát biểu: Cho tập P gồm n điểm nằm trên mặt phẳng:

$$P = \{p_1, p_2, \dots, p_n\}$$

Gọi $d(p_i, p_j)$ là khoảng cách Euclid giữa hai điểm p_i và p_j. Hãy tìm cặp (p_i, p_j) sao cho $d(p_i, p_j)$ là nhỏ nhất.

Với tập dữ liệu P trên mặt phẳng hai chiều, ta cho rằng các điểm trong P đã được sắp thứ tự không giảm theo hoành độ.

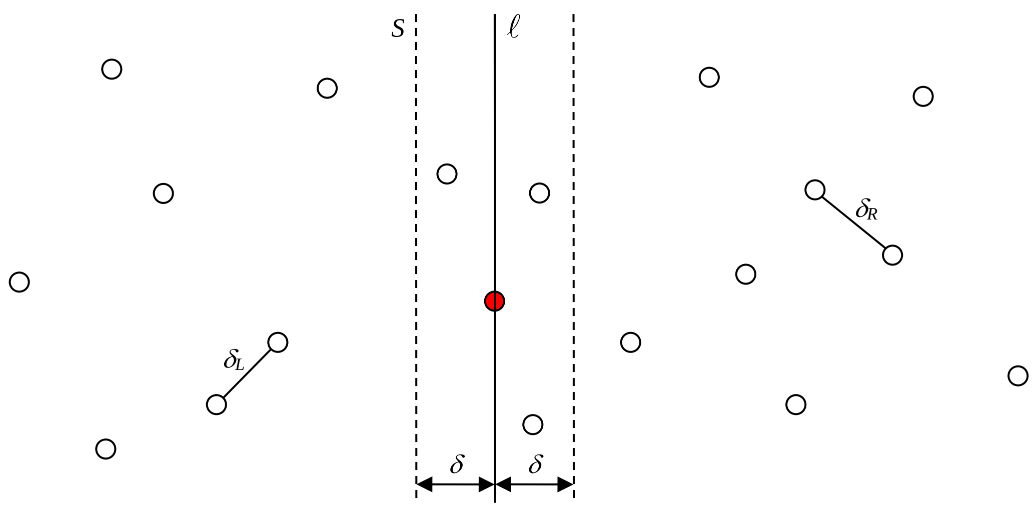
Nếu $2 \le n \le 3$ thì tính trực tiếp khoảng cách giữa các điểm này và đây được xem như là điều kiện để dừng quá trình đệ qui.

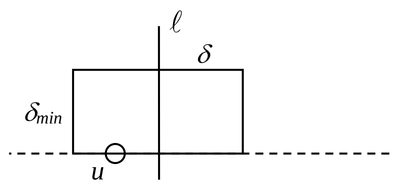
Với $n > 3$:

– Gọi ℓ là đường thẳng song song với trục tung và đi qua một điểm trong P có hoành độ là trung vị. Đường này phân chia tập P thành hai tập điểm P_L (nằm bên trái) chứa $\left\lceil \frac{n}{2} \right\rceil$ điểm và P_R (nằm bên phải) chứa $\left\lfloor \frac{n}{2} \right\rfloor$ điểm.

– Tìm cặp điểm gần nhất (một cách đệ qui) trong tập P_L (khoảng cách δ_L) và P_R (khoảng cách δ_R). Gọi $\delta = \min\{\delta_L, \delta_R\}$.

– δ có thể chưa phải là khoảng cách ngắn nhất. Một cặp điểm (u, v) có khoảng cách ngắn hơn δ có thể tìm thấy với $u \in P_L$ và $v \in P_R$ (hoặc ngược lại).





Giải thuật

```
ClosestPair(Point P[1..n]) {
   Q = P;
   Sắp xếp tập điểm P theo thứ tự hoành độ không giảm;
   Sắp xếp tập điểm Q theo thứ tự tung độ không giảm;
   δ = ClosestPairRec(P, Q);
}

ClosestPairRec(Point P[1..n], Point Q[1..n]) {
   if (|P| ≤ 3)
      Tìm cặp nhỏ nhất theo cách thông thường và trả về;

   ℓ = P[⌈n/2⌉].x // Giá trị trung vị, theo hoành độ
   Sao chép ⌈n/2⌉ điểm đầu tiên trong P vào P_L;
   Sao chép cũng ⌈n/2⌉ điểm này trong Q vào Q_L, duy trì thứ tự;
   Sao chép ⌊n/2⌋ điểm còn lại trong P vào P_R;
   Sao chép cũng ⌊n/2⌋ điểm này trong Q vào Q_R, duy trì thứ tự;

   δ_L = ClosestPairRec(P_L, Q_L);
   δ_R = ClosestPairRec(P_R, Q_R);
   δ = min(δ_L, δ_R);

   Duyệt tuần tự Q, sao chép các điểm p thỏa |p.x - ℓ| < δ vào mảng S[1..k];
   δ_min = δ;
   for (i = 1; i < k; i++) {  // S[i] ≡ u
      j = i + 1;              // S[j] ≡ v
      while (j ≤ k) && (|S[i].y - S[j].y| < δ_min) {
         tmp = sqrt((S[i].x - S[j].x)^2 + (S[i].y - S[j].y)^2);
         if (tmp < δ_min)
            δ_min = tmp;
         j++;
      }
   }
   return  δ_min;
}
```

Đánh giá: Hệ thức truy hồi là

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

Bài toán đổi tiền xu

Phát biểu: Giả sử có các mệnh giá tiền xu là $x_1, x_2, \ldots, x_k$. Tìm số lượng đồng tiền xu nhỏ nhất để có thể đổi n xu.

Giải thuật

```
moneyChange(coins[1..k], money) {
   for (i = 1; i ≤ k; i++)
      if (coins[i] == money)
         return   1;
   minCoins = money;
   for (i = 1; i ≤ money / 2; i++) {
      tmpSum = moneyChange(coins, i) + moneyChange(coins, money - i);
      if (tmpSum < minCoins)
         minCoins = tmpSum;
   }
   return   minCoins;
}
```

Đánh giá: Hệ thức truy hồi chia để trị là:

$$T(n) = \begin{cases} \sum_{i=1}^{\lfloor n/2 \rfloor} \big(T(i) + T(n-i)\big) + \Theta\left(\frac{n}{2}\right) & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

$$T(n) \in \Omega(2^n)$$

Giải thuật (cải tiến)

```
moneyChange(coins[1..k], money) {
   for (i = 1; i ≤ k; i++)
      if (coins[i] == money)
         return   1;
   minCoins = money;
   for (i = 1; i ≤ k; i++)
      if (money > coins[i]) {
         tmpSum = 1 + moneyChange(coins, money - coins[i]);
         if (tmpSum < minCoins)
            minCoins = tmpSum;
      }
   return   minCoins;
}
```

Kỹ thuật Giảm để trị (Decrease-and-Conquer)

Giới thiệu chung

Giảm với lượng không đổi (*decrease by a constant*)
Kích thước dữ liệu giảm một lượng không đổi sau mỗi bước lặp, thường là 1.

Ví dụ: Tính a^n.

$$f(n) = \begin{cases} f(n-1) \times a & n > 1 \\ a & n = 1 \end{cases}$$

Ví dụ: Sắp xếp chèn với dãy số $a_1, a_2, \dots, a_n$.

Giảm với tỉ lệ không đổi (*decrease by a constant factor*)
Kích thước dữ liệu giảm bởi tỉ lệ α (thường là $\frac{1}{2}$) không đổi sau mỗi bước lặp.

Ví dụ: Tìm kiếm nhị phân. Lời giải với dữ liệu kích thước n cũng vẫn là lời giải với dữ liệu kích thước $\frac{n}{2}$.

Ví dụ: Tính a^n.

$$a^n = \begin{cases} (a^{n/2})^2 & n \equiv_2 0 \wedge n \neq 0 \\ \left(a^{\lfloor n/2 \rfloor}\right)^2 \times a & n \equiv_2 1 \\ 1 & n = 0 \end{cases}$$

Ví dụ: Cách nhân của tá điền Nga

$$n \times m = \begin{cases} \dfrac{n}{2} \times 2m & n \equiv_2 0 \\ \left\lfloor \dfrac{n}{2} \right\rfloor \times 2m + m & n \equiv_2 1 \end{cases}$$

Giảm với lượng biến đổi (*variable size decrease*)
Kích thước của thể hiện bài toán giảm với lượng không xác định sau mỗi lượt lặp.

Ví dụ: Giải thuật Euclid để tìm USCLN của hai số a và b.

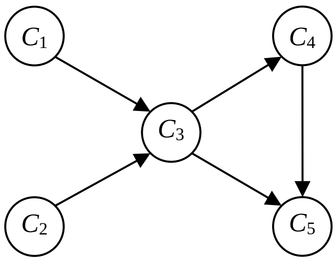
$$USCLN(a, b) = USCLN(b, a \,\%\, b)$$

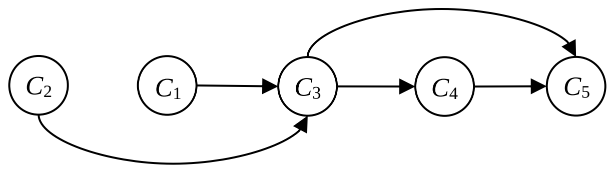
Ví dụ: Tìm một phần tử trên cây nhị phân tìm kiếm.

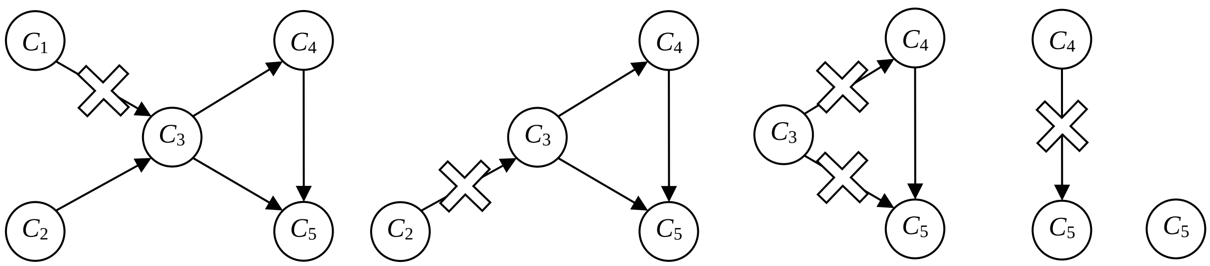
Kỹ thuật Giảm với lượng không đổi

Sắp xếp Topo

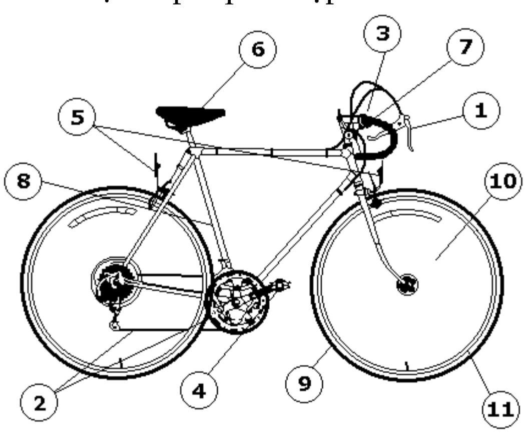
Ví dụ: Gọi $\{C_1, C_2, \ldots, C_5\}$ là tên của 5 môn học mà một sinh viên bắt buộc phải hoàn thành trong khóa học ngắn hạn nào đó.

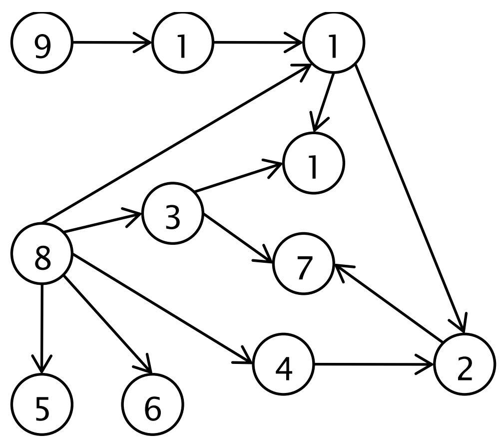




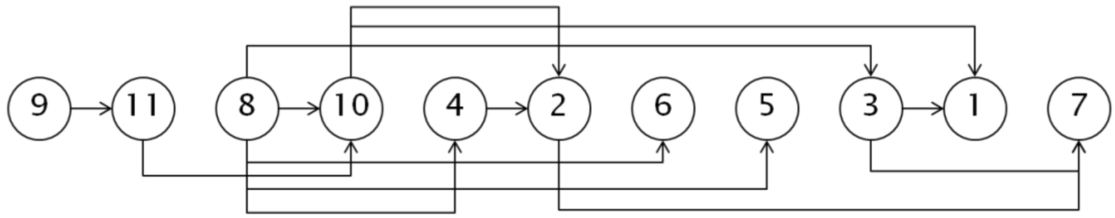


Ví dụ: Lắp ráp xe đạp





1. Lắp thắng vào tay lái
2. Gắn bộ truyền động
3. Gắn tay lái
4. Lắp bàn đạp và đĩa
5. Lắp đèn xe
6. Lắp yên xe
7. Lắp hộp điều chỉnh tốc độ
8. Lắp (tạo) khung xe
9. Lắp vỏ xe vào vành xe
10. Gắn bánh xe vào khung xe
11. Lắp vành xe vào bánh xe (đúc)



Giải thuật thô:

Bước 1: Duyệt mảng `indegree` và tìm ra những đỉnh có bậc vào bằng 0. Những đỉnh này sẽ được đưa lần lượt vào hàng chờ `Q`.

Bước 2: *Nếu* hàng chờ `Q` khác rỗng thì lấy đỉnh *u* ra khỏi hàng, in ra (lần lượt từ *trái sang phải*) và chuyển sang *Bước 3*;

Ngược lại: Dừng.

Bước 3: Với mỗi đỉnh *v* kề với đỉnh *u*, tiến hành giảm *bậc vào* của đỉnh *v* trong mảng `indegree` (tương đương việc loại các cạnh từ *u* đến *v*). Nếu phát hiện *bậc vào* của đỉnh *v* nào đó trở về 0 thì thêm vào hàng chờ `Q`. Quay về *Bước 2*.

Giải thuật:

```
TopologicalSort(Graph G) {
   indegree[1 .. |V|] = {0};
   priorQueue  Q = ∅;

   for (mỗi u ∈ V)
      for (mỗi đỉnh v kề với u)
         indegree[v] ++;
   for (mỗi v ∈ V)
      if (indegree[v] == 0)
         enQueue(Q, v);

   while (!isEmpty(Q)) {
      Vertex u = deQueue(Q);
      Output(u);
      for (mỗi đỉnh v kề với u)
         if( -- indegree[v] == 0 )
            enQueue(Q, v);
   }
}
```

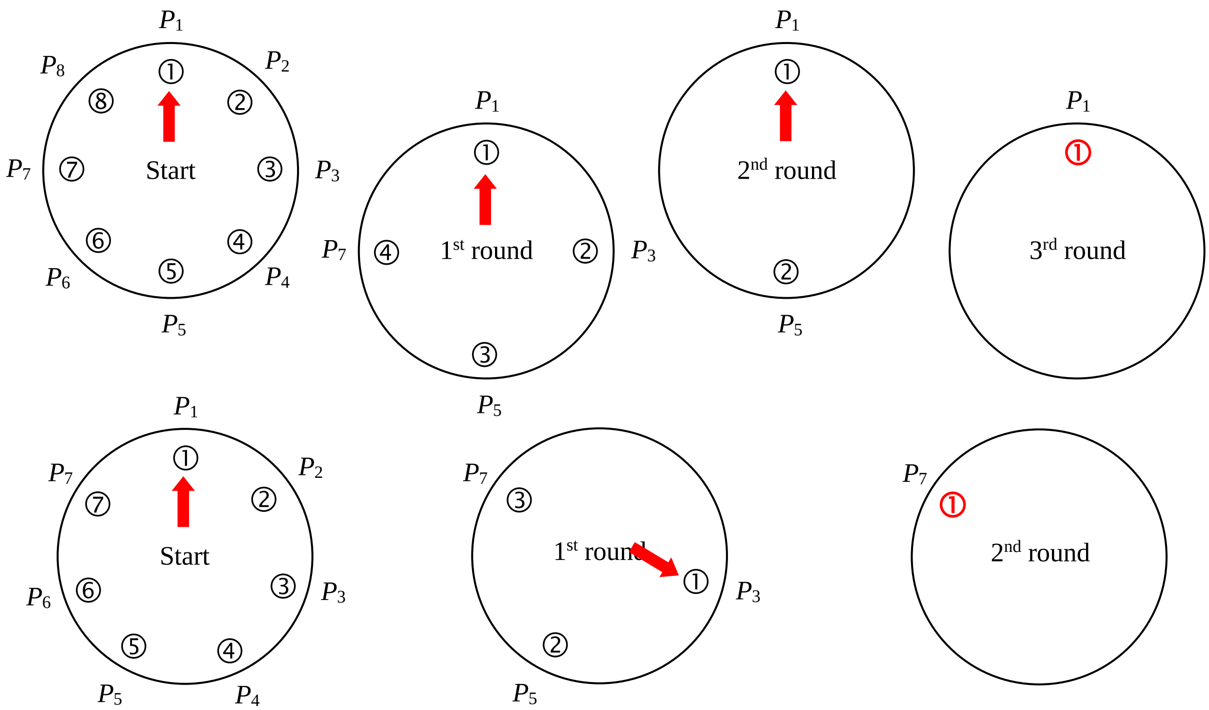
Đánh giá: Nếu đầu vào của giải thuật là danh sách kề thì chi phí của nó là $\Theta(|V| + |E|)$.

Kỹ thuật giảm với tỉ lệ không đổi

Bài toán Josephus

Phát biểu: Có n người đứng thành vòng tròn. Bắt đầu từ người có số thứ tự 1, người kế bên sẽ bị loại bỏ cho đến khi chỉ còn một người. *Hãy xác định số thứ tự $J(n)$ ban đầu của người sẽ trụ lại cuối cùng*.

Ví dụ: $J(8) = 1, J(7) = 7, J(6) = 5$.



Để xác định vị trí $J(n)$, chúng ta chia ra hai trường hợp:

- $n = 2h$:

$$J(2h) = 2J(h) - 1$$

- $n = 2h + 1$:

$$J(2h + 1) = 2J(h) + 1$$

Công thức tổng quát là:

$$J(2^k + i) = 2i + 1, i \in [0, 2^k - 1]$$

Kỹ thuật giảm với lượng biến đổi

Tìm USCLN(a, b) – Giải thuật Euclid

Phát biểu: Gọi $a, b \in \mathbb{Z}^{+}$. Tìm USCLN của hai số này.

Định lý (của) Lame: Gọi $a, b \in \mathbb{Z}^{+}$, với $a \geq b \geq 2$. Số lượng các phép chia nguyên của giải thuật Euclid để tìm ra $USCLN(a, b)$ không vượt quá 5 lần chiều dài của số nguyên b.

Giải thuật

```
gcd(a, b) {
   while (b) {
      t = b;       // a sẽ nhận giá trị này
      b = a % b;   // b sẽ nhận giá trị mới, nhỏ hơn
      a = t;
   }
   return   a;
}
```

Cây nhị phân tìm kiếm

Phát biểu: Xác định sự hiện diện của giá trị *k* trên cây nhị phân tìm kiếm.

Đánh giá:

- Trường hợp tốt nhất: $\Theta(\log n)$
- Trường hợp xấu nhất: $\Theta(n)$
- Trường hợp trung bình: $\Theta(\log n)$

Bài toán chọn (Selection problem)

Phát biểu: Tìm phần tử *nhỏ thứ k* trong dãy $S = \{s_1, s_2, \dots, s_n\}$, với $k \in [1, n]$.

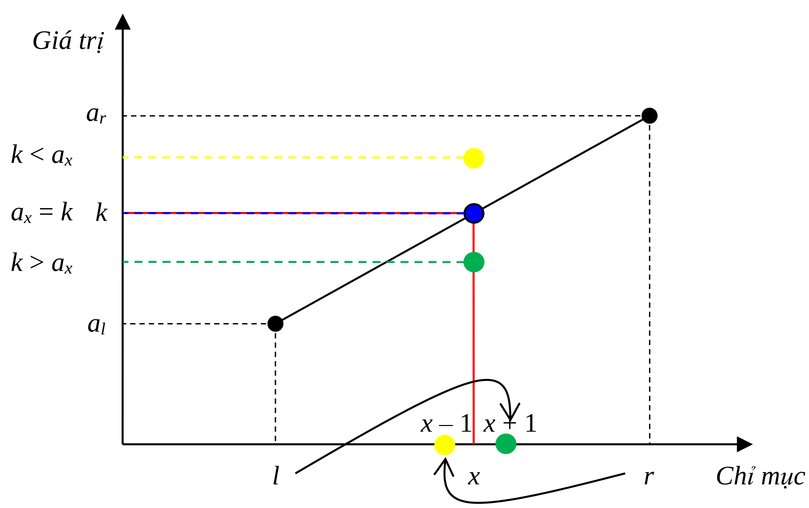
Đánh giá:

- Trường hợp tốt nhất: $\Theta(n)$
- Trường hợp xấu nhất: $O(n^2)$
- Trường hợp trung bình: $\Theta(n)$

Giải thuật

```
Selection(S[], lower, upper, k) {
   h = random(lower, upper);
   pos = Partition(S, lower, upper, h);    // S[pos] = pivot
   if (pos == k)
      return S[pos];
   if (pos > k)
      Selection(S, lower, pos - 1, k);
   if (pos < k)
      Selection(S, pos + 1, upper, k);
}
Partition(S, lower, upper, pos) {
   pivot = S[pos];
   S[lower] ⇆ S[pos];
   pos = lower;
   for (i = lower + 1; i ≤ upper; i++)
      if (pivot > S[i]) {
         pos++;
         S[i] ⇆ S[pos];
      }
   S[lower] ⇆ S[pos];
   return   pos;
}
```

Tìm kiếm nội suy



Nếu gọi k là giá trị cần tìm ($a_l \le k \le a_r$) thì vị trí x "tương ứng với k" được xác định bởi công thức sau:

$$\frac{x-l}{r-l} = \frac{k-a_l}{a_r-a_l} \Longrightarrow x = l + \left\lfloor \frac{(k-a_l)(r-l)}{a_r-a_l} \right\rfloor$$

Có được giá trị x, chúng ta hy vọng a_x sẽ là phần tử cần tìm:

- Nếu $a_x = k$: Dừng tìm kiếm.
- Nếu $a_x > k$: Giải thuật tiếp tục tìm kiếm trong đoạn l và $(r =)x - 1$.
- Nếu $a_x < k$: Đoạn tìm kiếm kế tiếp là $(l =)x + 1$ và r.

Qui hoạch động

Nguyên lý của sự tối ưu

"Lời giải tối ưu cho thực thể bất kỳ của bài toán tối ưu được hình thành từ việc tổ hợp các lời giải tối ưu cho những thực thể nhỏ hơn".

Kỹ thuật này hướng đến việc giải quyết những vấn đề mang bản chất đệ qui chia để trị, khi tồn tại quan hệ truy hồi giữa lời giải của bài toán ban đầu và lời giải của các bài toán con cùng kiểu.

Ví dụ: Tìm số thứ n của dãy Fibonacci.

Ví dụ: Tính hệ số nhị thức $\binom{n}{k}$.

Hệ số nhị thức được xác định bởi công thức sau:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}\ với\ 0 \le k \le n$$

và tính đệ qui được thể hiện bởi công thức:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \lor k = n \end{cases}$$

Công thức tổng quát là:

$$\binom{i}{j} \Leftrightarrow C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & 0 < j < i \\ 1 & j = 0 \lor j = i \end{cases}$$

	0	1	2	…	$j-1$	j	…	$k-1$	k
0	1								
1	1	1							
2	1	2	1						
⋮									
$i-1$					$C[i–1, j–1]$	$C[i–1, j]$			
i						$C[i, j]$			
⋮									
k	1								1
⋮									
$n–1$	1							$C[n–1, k–1]$	$C[n–1, k]$
n	1								$C[n, k]$

Giải thuật (đệ qui)

```
Binomial(n, k) {
   if (k == 0 || k == n)
      return 1;
   return Binomial(n - 1, k - 1) + Binomial(n - 1, k);
}
```

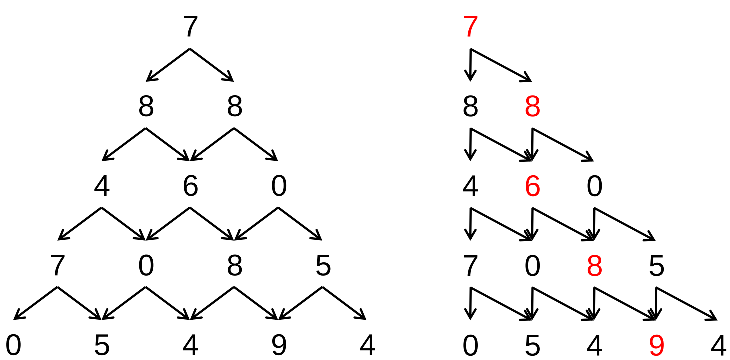
Giải thuật

```
Binomial(n, k) {
   C[0 .. n, 0] = C[0 .. k, 0 .. k] = 1;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j < (i ≤ k ? i : k + 1); j++)
         C[i, j] = C[i - 1, j - 1] + C[i - 1, j];
   return   C[n, k];
}
```

Đánh giá: $A(n, k) \in \Theta(nk)$

Ví dụ: Bài toán tam giác

Phát biểu: Cho trước tam giác "đều" cạnh n chứa các số nguyên dương. Tính tổng lớn nhất của một con đường đi từ đỉnh đến đáy và chỉ ra con đường này. Yêu cầu là mỗi bước đi xuống lệch sang trái hoặc phải một vị trí).



(a) (b)

	0	1	2	3	4	5
0	0	0				
1	0	7	0			
2	0	15	15	0		
3	0	19	21	15	0	
4	0	26	21	29	20	0
5	0	26	31	33	38	24

(c)

Gọi $S(i, j)$ là tổng lớn nhất của đoạn đường từ đỉnh đến vị trí dòng i và cột j.

$$S(i,j) = \begin{cases} \max\big(S(i-1,j-1), S(i-1,j)\big) + a_{i,j} & i \neq j \wedge i,j \neq 1 \\ a_{i,j} & i = j = 1 \\ S(i-1,j-1) + a_{i,j} & i = j \wedge i,j \neq 1 \\ S(i-1,j) + a_{i,j} & i \neq j \wedge j = 1 \end{cases}$$

Sử dụng mảng S hai chiều $(n + 1) \times (n + 1)$ để tính toán mọi con đường có thể xuất phát từ đỉnh. Ban đầu, $S[0..n, 0] = 0, S[j, j + 1] = 0$ với $0 \leq j \leq n - 1$.

Giải thuật

```
… Đưa mảng a vào bảng S …

// Xây dựng bảng S
S[0 .. n, 0] = S[0 .. n − 1, 1 .. n] = 0;
for (i = 1; i ≤ n; i++)
   for (j = 1; j ≤ i; j++)
      S[i][j] = max(S[i - 1][j - 1], S[i-1][j]) + a[i][j];
return   "Phần tử lớn nhất trên dòng n"
```

Đánh giá: $S(n) \in \Theta(n^2)$

Mở rộng: Xác định con đường dẫn đến kết quả

Bài toán đổi tiền xu

Gọi mệnh giá k đồng xu là $\{x_1, x_2, \dots, x_k\}$ và giả định rằng, đồng xu có mệnh giá nhỏ nhất là 1 xu để đảm bảo luôn có lời giải.

Gọi $C(n)$ là số lượng đồng xu tối thiểu để đổi n xu:

$$C(n) = 1 + \min_{1 \le i \le k} \{C(n - x_i)\} \text{ với } n \ge x_i, C(0) = 0$$

Giải thuật

```
int   changeCoinsDP(int coins[], int k, int money) {
   int C[0 .. money] = 0;

   for (cents = 1; cents ≤ money; cents++) {
      int minCoins = cents;
      for (i = 1; i ≤ k; i++) {
         if (coins[i] > cents)
            continue;
         if (C[cents - coins[i]] + 1 < minCoins)
            minCoins = C[cents - coins[i]] + 1;
      }
      C[cents] = minCoins;
   }
   return C[money];
}
```

Đánh giá: $\Theta(kn)$

Mở rộng: Xác định các đồng xu trong lời giải.

Tìm dãy con tăng nghiêm ngặt dài nhất

Phát biểu: Cho dãy số (nguyên dương) $a_1, a_2, \dots, a_n$. Một dãy con tăng nghiêm ngặt được hình thành bằng cách loại bỏ không hoặc nhiều phần tử nào đó để những phần tử còn lại $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ với $1 \le k \le n$ thỏa điều kiện $a_{i_p} < a_{i_q}$ với $1 \le p < q \le k$. Tìm dãy con tăng nghiêm ngặt dài nhất.

Gọi $L(i)$ là chiều dài của dãy con tăng nghiêm ngặt dài nhất, *với a_i là phần tử cuối cùng trong dãy này và tất nhiên có giá trị lớn nhất trong dãy*.

$$L(i) = 1 + \max_{1 \le j < i:\, a_j < a_i} \{L(j)\}$$

với $L(1) = 1$

Giải thuật (đệ qui)

```
lis(a[1 .. n], i) {
   if (i == 1)
      return 1;

   tmpMax = 1;
   for (j = 1; j < i; j++)
      if (a[j] < a[i]) {
         res = lis(a, j);
         if (res + 1 > tmpMax)
            tmpMax = res + 1;
      }
   if (max < tmpMax) // global variable
      max = tmpMax;
   return   tmpMax;
}

for (i = 1; i ≤ n; i++)
   lis(a, i);
print(max);
```

Đánh giá: Độ phức tạp của giải thuật phụ thuộc vào phân bố của dữ liệu đầu vào. Để đơn giản, cho rằng các phần tử dữ liệu khác nhau từng đôi một. Gọi $T(i)$ là chi phí thực thi của hàm `lis(a, i)`, $\forall i \in [1, n]$.

- Trường hợp xấu nhất: $\Theta(2^n)$
- Trường hợp tốt nhất: $\Theta(n^2)$
- Trường hợp trung bình: Không dễ để xác định do phụ thuộc vào kết quả của phép so sánh cơ sở.

Với tiếp cận qui hoạch động, gọi $L[i]$ là chiều dài dài nhất của dãy con tăng nghiêm ngặt có a_i là phần tử cuối cùng (thuộc về dãy con này). Nhận thấy:

$$L[i] = \max_{1 \le j < i:\, a_j < a_i} \{L[j]\} + 1$$

Giải thuật

```
lis_DP(a[1..n]) {
   L[1 .. n] = 1;
   for (i = 2; i ≤ n; i++)
      for (j = 1; j < i; j++)
         if ((a[j] < a[i]) && (L[j] + 1 > L[i]))
            L[i] = L[j] + 1;
   return   "Phần tử lớn nhất trên mảng L";
}
cout << lis_DP(a);
```

Đánh giá: $\Theta(n^2)$

Mở rộng: Chỉ ra dãy con tăng tuyệt đối dài nhất.

Tìm dãy con chung dài nhất

Phát biểu: Cho hai chuỗi ký tự $S = s_1 s_2 \dots s_m$ và $T = t_1 t_2 \dots t_n$. Tìm dãy con chung dài nhất (và chiều dài của nó) xuất hiện trong cả hai chuỗi trên. Nói cách khác, đây là dãy các vị trí trong S: $1 \le i_1 < i_2 < \cdots < i_k \le m$ và trong T: $1 \le j_1 < j_2 < \cdots < j_k \le n$, sao cho ký tự $s_{i_h} \equiv t_{j_h}, \forall h \in [1, k]$ và k là lớn nhất.

Ví dụ: $S = XYXZPQ, T = YXQYXP$. Dãy con chung dài nhất là $XYXP$ với độ dài 4, vì $S = XYXZPQ, T = YXQYXP$.

Gọi $L(i, j)$ là chiều dài của *dãy con chung dài nhất* nằm bên trong hai *chuỗi con* $s_1 s_2 \dots s_i$ và $t_1 t_2 \dots t_j$.

- Nếu $s_i = t_j$: $L(i, j) = 1 + L(i-1, j-1)$
- Nếu $s_i \ne t_j$: $L(i, j) = \max\{L(i-1, j), L(i, j-1)\}$

với $L(i, 0) = L(0, j) = 0$.

Giải thuật (đệ qui)

```
int   LCS(char S[], int i, char T[], int j) {
   if ((i == 0) || (j == 0))
      return 0;

   if (S[i] == T[j])
      return   1 + LCS(S, i - 1, T, j - 1);
   else
      return   max(LCS(S, i - 1, T, j), LCS(S, i, T, j - 1));
}
cout << LCS(S, m, T, n);
```

Đánh giá: $\Theta(2^n)$

Với qui hoạch động, sử dụng bảng hai chiều L kích thước $m \times n$ để lưu trữ kết quả của những thực thể của bài toán con.

Ô $L[i,j]$ chứa chiều dài của *dãy con chung dài nhất* nằm trên hai *chuỗi con* $s_1 s_2 \dots s_i$ và $t_1 t_2 \dots t_j$.

- Nếu $s_i = t_j$: $L[i,j] = L[i-1,j-1] + 1$
- Nếu $s_i \neq t_j$: $L[i,j] = \max\{L[i-1,j], L[i,j-1]\}$

với $L[0,j] = L[i,0] = 0$. Rõ ràng, $L[m,n]$ chứa chiều dài cần tìm.

Giải thuật

```
LCS_Dyn(char S[], int m, char T[], int n) {
   L[1 .. m][0] = L[0][1 .. n] = 0

   for (i = 1; i ≤ m; i++)
      for (j = 1; j ≤ n; j++)
         if (S[i] == T[j])
            L[i][j] = 1 + L[i - 1][j - 1];
         else
            L[i][j] = max(L[i - 1][j], L[i][j - 1]);
   return   L[m][n];
}
```

Đánh giá: Chi phí của giải thuật là $\Theta(mn)$.

Mở rộng: Tìm dãy con chung.

Giải thuật (của) Floyd tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh

Phát biểu: Cho đồ thị *liên thông* có trọng số (có hướng hay vô hướng). Tìm khoảng cách ngắn nhất từ mỗi đỉnh đến mọi đỉnh còn lại.

Cho rằng đồ thị có n đỉnh. Sử dụng mảng hai chiều D (gọi là *ma trận khoảng cách*) kích thước $n \times n$ với phần tử d_{ij} là chiều dài ngắn nhất đi từ đỉnh nhãn i đến đỉnh nhãn j ($1 \leq i \neq j \leq n$).

Giải thuật (của) Floyd sẽ tính toán ma trận D thông qua dãy ma trận:

$$D^{(0)}, D^{(1)}, \dots, D^{(k-1)}, D^{(k)}, \dots, D^{(n)} \equiv D$$

Phần tử $d_{ij}^{(k)} \in D^{(k)}$ (với $k = 0,1, \dots, n$) là chiều dài con đường ngắn nhất (trong tất cả các con đường) đi từ đỉnh nhãn i đến đỉnh nhãn j sao cho, các đỉnh trung gian (nếu có) được đánh số từ k trở xuống.

$$d_{ij}^{(k)} = \min_{1 \leq k \leq n} \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\}$$

Giải thuật

```
Floyd(W[1 .. n, 1 .. n]) { // Ma trận kề chứa trọng số
   D(0) = W;
   for (k = 1; k ≤ n; k++)
      for (i = 1; i ≤ n; i++)
         for (j = 1; j ≤ n; j++)
            D(k)[i, j] = min{D(k-1)[i, j], D(k-1)[i, k] + D(k-1)[k, j]};
   return   D(n);
}
```

Đánh giá: $\Theta(n^3)$

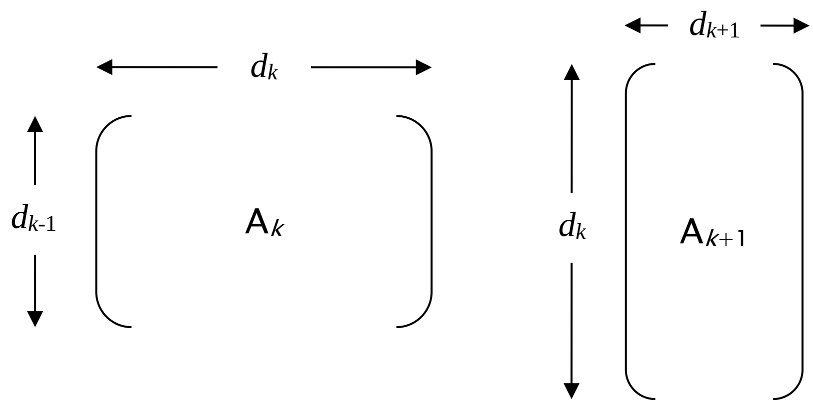
Nhân dãy ma trận

Phát biểu: Xác định thứ tự để nhân dãy n ma trận $A_1 \times A_2 \times \ldots \times A_n$ có kích thước $d_0 \times d_1, d_1 \times d_2, \ldots, d_{n-1} \times d_n$ với chi phí thấp nhất.

Ví dụ: Nhân 4 ma trận $A \times B \times C \times D$ với kích thước lần lượt là $50 \times 20, 20 \times 1, 1 \times 10, 10 \times 100$. So sánh *ba* trong số *năm* thứ tự nhân 4 ma trận nêu trên:

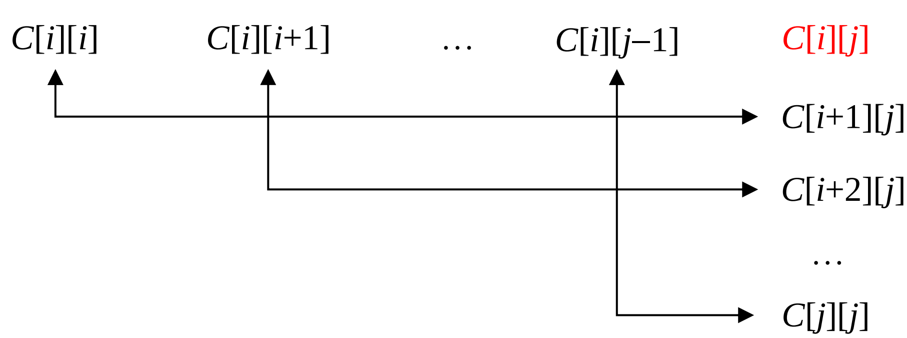
Thứ tự nhân	Số phép tính
$A \times \big((B \times C) \times D\big)$	$20 \times 1 \times 10 + 20 \times 10 \times 100 + 50 \times 20 \times 100 = 120200$
$\big(A \times (B \times C)\big) \times D$	$20 \times 1 \times 10 + 50 \times 20 \times 10 + 50 \times 10 \times 100 = 60200$
$(A \times B) \times (C \times D)$	$50 \times 20 \times 1 + 1 \times 10 \times 100 + 50 \times 1 \times 100 = 7000$

Qui ước: Nếu $A_k \times A_{k+1}$ với $1 \le k < n$ thì kích thước của ma trận A_k và A_{k+1} lần lượt là $d_{k-1} \times d_k$ và $d_k \times d_{k+1}$ và kích thước ma trận tích là $d_{k-1} \times d_{k+1}$.



Xét tích $A_i \times A_{i+1} \times \ldots \times A_j$ với $1 \le i \le j \le n$. Gọi $C(i,j)$ là chi phí tối thiểu của dãy phép nhân này.

$$C(i,j) = \begin{cases} \min\limits_{i \le k < j} \{C(i,k) + C(k+1,j) + d_{i-1} \times d_k \times d_j\} & i < j \\ 0 & i = j \end{cases}$$



Giải thuật

```
ChainMatrixMult(d[0 .. n], P[1 .. n][1 .. n]) {
   C[1 .. n, 1 .. n] = 0;

   for (diag = 1; diag < n; diag++)
      for (i = 1; i ≤ n - diag; i++) {
         j = i + diag;
         C[i, j] = min_{i≤k<j}{C[i,k] + C[k+1,j] + d[i-1] × d[k] × d[j]};
         P[i, j] = Gia tri k tim duoc;
      }

   return C[1, n];
}
```

Đánh giá: $\Theta(n^3)$

Mở rộng:

Mảng P hai chiều có nhiệm vụ lưu lại giá trị phân tách k khiến cho tích của dãy ma trận từ A_i đến A_j là nhỏ nhất.

Giải thuật

```
order(i, j) {
   if (i == j)
      cout << "A" <<  i;
   else {
      k = P[i][j];
      cout << "(";
      order(i, k);
      order(k + 1, j);
      cout << ")";
   }
}

order(1, n);
```

Cây nhị phân tìm kiếm tối ưu

Cây nhị phân tìm kiếm cổ điển xem các khóa tìm kiếm là như nhau. Nếu thu thập được thông tin về tần suất mỗi khóa được tìm kiếm thì để tăng hiệu quả truy xuất:

- Đặt những khóa được tìm kiếm nhiều ở gần nút gốc,
- Đặt những khóa càng ít tìm càng nằm xa.

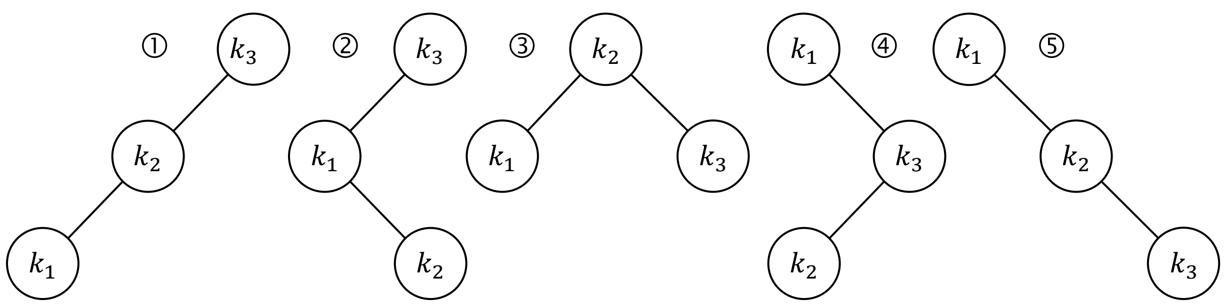
Như vậy, chi phí trung bình cho việc tìm kiếm sẽ là tối thiểu. Cây có tính chất này gọi là Cây nhị phân tìm kiếm tối ưu.

Qui ước: Gọi

- $k_1, k_2, \dots, k_n$ là khóa của n nút trên cây. Cho rằng $k_1 < k_2 < \cdots < k_n$
- p_i là xác suất để k_i trở thành khóa tìm kiếm
- c_i là số phép so sánh để tìm ra k_i:
$$c_i = level(k_i) + 1$$
- Chi phí chung để tìm một khóa bất kỳ trên cây là
$$Cost = \sum_{i=1}^{n} (c_i \times p_i)$$

và chúng ta cần xây dựng cây sao cho giá trị này là nhỏ nhất.

Ví dụ: Xét cây nhị phân tìm kiếm có ba nút, với các xác suất lần lượt là $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$. Khóa các nút là: $k_1 < k_2 < k_3$. Có 5 khả năng hình thành cây:



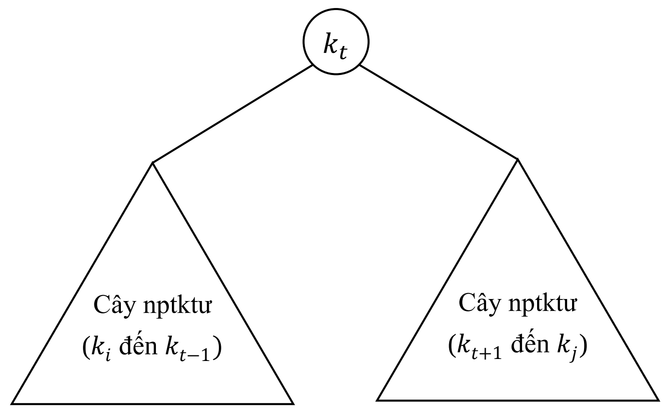
1. 3 (0.7) + 2 (0.2) + 1 (0.1) = 2.6
2. 2 (0.7) + 3 (0.2) + 1 (0.1) = 2.1
3. 2 (0.7) + 1 (0.2) + 2 (0.1) = 1.8
4. 1 (0.7) + 3 (0.2) + 2 (0.1) = 1.5
5. 1 (0.7) + 2 (0.2) + 3 (0.1) = **1.4**

Gọi $C(i,j)$ là chi phí chung nhỏ nhất để tìm một khóa bất kỳ trên cây nhị phân tìm kiếm chứa các khóa từ k_i đến k_j. Cây nhị phân này được gọi là $T_{i,j}$ với $1 \leq i \leq j \leq n$.

- Nếu $i = j$: Cây $T_{i,j}$ chỉ có một nút.

$$C(i,i) = c_i \times p_i = 1 \times p_i = p_i$$

- Nếu $i > j$: Cây $T_{i,j}$ được xem là rỗng và $C(i,j) = 0$.
- Nếu $i < j$: Quan tâm đến mọi khả năng có thể để hình thành nên một cây $T_{i,j}$. Gọi $T_{i,j}^t$ là cây $T_{i,j}$ có gốc chứa khóa k_t nào đó với $i \leq t \leq j$.



$$C(i,t-1) = \sum_{s=i}^{t-1} (\#\ So\ sánh\ tìm\ key_s) \times p_s = \sum_{s=i}^{t-1} c_s \times p_s$$

$$C(t+1,j) = \sum_{s=t+1}^{j} (\#\ So\ sánh\ tìm\ key_s) \times p_s = \sum_{s=t+1}^{j} c_s \times p_s$$

Cây $T_{i,j}^t$ có chi phí tìm kiếm chung là:

$$\left(C(i,t-1) + \sum_{s=i}^{t-1} p_s\right) + \left(C(t+1,j) + \sum_{s=t+1}^{j} p_s\right) + p_t$$

$$= C(i,t-1) + C(t+1,j) + \sum_{s=i}^{j} p_s$$

$$\Longrightarrow C(i,j) = \min_{i \leq t \leq j} \left\{ C(i,t-1) + C(t+1,j) + \sum_{s=i}^{j} p_s \right\}$$

$$= \min_{i \leq t \leq j} \{C(i,t-1) + C(t+1,j)\} + \sum_{s=i}^{j} p_s$$

với $1 \leq i \leq j \leq n$.

Ví dụ: Xét bảng C kích thước $(1..n+1) \times (0..n)$ với $n = 10$.

$i\downarrow$ \ $j\rightarrow$	0	1	2	3	4	5	6	7	8	9	10
1	0	p_1									?
2		0	p_2								
3			0	p_3							
4				0	p_4			$C(4,7)$			
5					0	p_5					
6						0	p_6				
7							0	p_7			
8								0	p_8		
9									0	p_9	
10										0	p_{10}
11											0

Giải thuật

```
OptimalBST(p[1..n]) {
   int C[1 .. n + 1, 0 .. n], R[1 .. n + 1, 0 .. n], ;
   C[1 .. n + 1, 0 .. n] = R[1 .. n + 1, 0 .. n] = 0;
   C[1 .. n, 1 .. n] = p[1 .. n];
   R[1 .. n, 1 .. n] = 1 .. n;
   for (diag = 1; diag < n; diag++)
      for (i = 1; i ≤ n – diag; i++) {
         j = i + diag;
         minval = min_{i≤t≤j}(C[i, t − 1] + C[t + 1, j]);
         R[i, j] = Giá trị t tìm được;
         C[i, j] = minval + ∑_{s=i}^{j} p[s];
      }
   return      <C[1, n], R>;
}
```

Đánh giá: $\Theta(n^3)$

Giải thuật (Xây dựng cây)

```
tree(i, j) {
   t = R[i, j];
   if (t == 0)     return NULL;
   p         = new node;
   p->key    = key[t];
   p->left   = tree(i, t - 1);
   p->right  = tree(t + 1, j);
   return p;
}
root = tree(1, n);
```

Tổng các tập con

Phát biểu: Tìm tập con của tập đã cho $A = \{a_1, a_2, \dots, a_n\}$ gồm n số nguyên dương sao cho tổng các phần tử của tập con này bằng với k.

Giả sử đã tồn tại hàm `SubsetSums(A, k)` kiểu `boolean` để tìm kiếm sự tồn tại của tập $S \subseteq A$ sao cho tổng các phần tử của tập con này bằng k. Chọn một phần tử bất kỳ trong A, gọi là a:

i. Nếu $a > k$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k)`.

ii. Ngược lại: câu trả lời sẽ là một trong hai khả năng sau:

a. Cho rằng phần tử $a \in S$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k-a)`.

b. Cho rằng phần tử $a \notin S$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k)`.

Điều kiện để quá trình đệ qui kết thúc là như sau:

– Nếu $k = 0$ (còn tập A có thể là rỗng hoặc không): Tồn tại tập S.
– Nếu $A = \emptyset$ và $k \neq 0$: Không tồn tại S.

Giải thuật (đệ qui)

```
SubsetSums(a[], n, k) {
   if (k == 0)
      return      true;
   if (n == 0)
      return      false;
   if (a[n] > k)
      return      SubsetSums(a, n - 1, k);
   return   SubsetSums(a, n - 1, k - a[n]) || SubsetSums(a, n - 1, k);
}
SubsetSums(a, n, k);
```

Đánh giá: $O(2^n)$

Với qui hoạch động, bảng $V[0..n, 0..k]$ sẽ được dùng để lưu giá trị:

- Nếu tồn tại *tập con* nào đó *của tập* $\{a_1, a_2, \dots, a_i\}$ có tổng là j $(1 \le j \le k)$:
$$V[i,j] = 1$$
- Ngược lại: $V[i,j] = 0$.

$$V[i,j] = \begin{cases} 1 & V[i-1,j] = 1 \lor V[i-1, j-a_i] = 1 \text{ với } j \ge a_i \\ 0 & \neq \end{cases}$$

với $V[0,1..k] = 0, V[0..n, 0] = 1$.

Giải thuật

```
SubsetSumsDP(a[1 .. n], n, k) {
   int V[0 .. n, 0 .. k];

   V[0 .. n, 0] = 1;
   V[0, 1 .. k] = 0;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j ≤ k; j++) {
         tmp = 0;
         if (j ≥ a[i])
            tmp = V[i - 1, j - a[i]];
         V[i, j] = V[i - 1, j] || tmp;
      }
}
SubsetSumsDP(a, n, k);
```

Đánh giá: Chi phí của giải thuật là $\Theta(nk)$.

Giải thuật (In kết quả)

```
if (V[n, k]) {
   while (k) {
      if (V[n - 1, k - a[n]] == 1 && V[n - 1, k] == 0) {
         cout << a[n] << " ";
         k -= a[n];
      }
      n--;
   }
}
```

Mở rộng 1

Phát biểu: Cho tập $A = \{a_1, a_2, \dots, a_n\}$ (có thể bằng nhau) gồm n số *nguyên dương*. Cho biết, tập này có thể chia thành hai tập con A_1 và A_2 sao cho: $A_1 \cap A_2 = \emptyset, A_1 \cup A_2 = A$ và tổng của các phần tử của hai tập *bằng nhau*.

Mở rộng 2

Phát biểu: Cho tập $A = \{a_1, a_2, \dots, a_n\}$ (có thể bằng nhau) gồm n số *nguyên dương*. Tìm cách chia tập này thành hai tập con A_1 và A_2 thỏa những điều kiện sau:

1. $A_1 \cap A_2 = \emptyset, A_1 \cup A_2 = A$
2. Nếu gọi S_A là tổng của các phần tử của tập A thì $\left|S_{A_1} - S_{A_2}\right|$ là nhỏ nhất.

Bài toán túi xách 0/1

Phát biểu: Cho n đồ vật có cân nặng là $w_1, w_2, \dots, w_n (\in \mathbb{Z}^+)$ và giá trị là $v_1, v_2, \dots, v_n (\in \mathbb{R}^+)$. Khả năng chứa của túi là $W (\in \mathbb{Z}^+)$. Tìm tập con có giá trị nhất của các đồ vật mà túi có thể mang được.

Gọi $T(i, j)$ là giá trị của tập con đáng giá nhất hình thành từ việc lấy một/nhiều/tất cả đồ vật trong số i đồ vật đầu tiên và cho *được* vào túi có sức chứa j.

$$T(i,j) = \begin{cases} \max\{T(i-1,j), v_i + T(i-1, j-w_i)\} & j \geq w_i \\ T(i-1,j) & j < w_i \end{cases}$$

$$\begin{cases} T(0,j) = 0 & j \geq 0 \\ T(i,0) = 0 & i \geq 0 \end{cases}$$

Ví dụ: Xét tập gồm 4 đồ vật: $\{w_1 = 2, v_1 = 12đ\}, \{w_2 = 1, v_2 = 10đ\}, \{w_3 = 3, v_3 = 20đ\}, \{w_4 = 2, v_4 = 15đ\}$ và $W = 5$.

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Giải thuật

```
Knapsack(w[1 .. n], v[1 .. n], W) {
   int T[0 .. n, 0 .. W];
   T[0 .. n, 0] = T[0, 1 .. W] = 0;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j ≤ W; j++)
         if (j ≥ w[i])
            T[i, j] = max{T[i - 1, j], v[i] + T[i - 1, j - w[i]]};
         else
            T[i, j] = T[i - 1, j];
   return   T[n, W];
}
```

Mở rộng: Xác định các thành phần của tập con.

Bài toán đường đi người bán hàng

Cho rằng, đồ thị liên thông $G = (V, E)$ có tập các đỉnh là $V = \{v_1, v_2, \dots, v_n\}$. Không mất đi tính tổng quát, gọi v_1 là đỉnh bắt đầu của mọi chu trình.

Nhận xét:

"Nếu v_i là đỉnh đầu tiên theo ngay sau v_1 trong một chu trình *tối ưu* thì đoạn đường còn lại của chu trình này, đi từ v_i quay về v_1, sẽ phải là con đường *ngắn nhất* đi qua mọi đỉnh còn lại đúng một lần".

Gọi:

- W: Ma trận kề biểu diễn G. Phần tử $W[i, j]$ có giá trị k (trọng số của cạnh nối từ v_i đến v_j), giá trị ∞ (không có cạnh nối) hoặc 0 (khi $i = j$).
- $A (\subseteq V)$: Tập các đỉnh.
- $D[v_i, A]$: Chiều dài của con đường ngắn nhất từ v_i về đến v_1, đi qua mọi đỉnh trong tập A đúng một lần.

Một cách tổng quát, $\forall i \in [2, n], v_i \notin A$:

$$D[v_i, A] = \begin{cases} \min\limits_{j: v_j \in A} \{W[i, j] + D[v_j, A \setminus \{v_j\}]\} & A \neq \emptyset \\ W[i, 1] & A = \emptyset \end{cases}$$

Giải thuật

```
TSP(n, W[1 .. n, 1 .. n], P[1 .. n, 1 .. n]) {
   D[1 .. n, Tập con của V \ {v1}];
   P[1 .. n, Tập con của V \ {v1}];

   D[2 .. n, ∅] = W[2 .. n, 1];
   for (k = 1; k ≤ n - 2; k++)       // Kích thước của tập A
      for (mỗi tập A (⊆ V \ {v1}) chứa k đỉnh)
         for (i: i ≠ 1 và vi ∉ A) {
            D[i, A] = min_{j:vj∈A} {W[i, j] + D[j, A \ {vj}]};
            P[i, A] = giá trị j tìm được;
         }
   D[1, V \ {v1}] = min_{2≤j≤n} {W[1][j] + D[j][V \ {v1, vj}]};
   P[1, V \ {v1}] = giá trị j tìm được;
   return   { D[1, V \ {v1}], P };
}
```

Đánh giá: $\Theta(n2^n)$

Mở rộng: Chỉ ra chu trình tối ưu

Giải thuật

```
TSP_Tour(P[1..n, 1..n]) {
   cout << v1;
   A = V \ {v1};
   k = 1;
   while (A ≠ ∅) {
      k = P[k, A];
      cout << vk;
      A = A \ {vk};
   }
}
```

Memoization

Một kỹ thuật lai giữa Chia để trị và Qui hoạch động để giải bài toán tối ưu. Cơ sở của kỹ thuật là sự kết hợp ưu điểm của hai tiếp cận truyền thống:

- Lối tư duy trực quan từ trên xuống (*top-down, depth-first analysis*) của Chia để trị thông qua cài đặt đệ qui.
- Tiếp cận từ dưới lên (*bottom-up, breadth-first analysis*) bằng cách lưu trữ kết quả các bài toán con của Qui hoạch động, sử dụng kỹ thuật lặp.

Ví dụ: Tìm số thứ n của dãy Fibonacci

Giải thuật

```
Fib(f[0 .. n], n) {
   if (f[n] < 0)
      f[n] = Fib(f, n - 1) + Fib(f, n - 2);
   return   f[n];
}
Fib_Memo(n) {
   f[0] = 0;
   f[1] = 1;
   f[2 .. n] = -1;
   return   Fib(f, n);
}
```

Ví dụ: Nhân dãy ma trận.

Hệ thức truy hồi của bài toán được chỉ ra như sau:

$$C(i,j) = \begin{cases} \min\limits_{i \le k < j} \{C(i,k) + C(k+1,j) + d_{i-1} \times d_k \times d_j\} & i < j \\ 0 & i = j \end{cases}$$

Giải thuật

```
ChainMatrixMult_Memo(d[0 .. n]) {
   C[.., ..] = ∞;
   return   MMM(C, d, 1, n);
}

MMM(C[1 .. n, 1 .. n], d[0 .. n], i, j) {
   if (C[i, j] < ∞)
      return   C[i, j];

   if (i == j)
      C[i, j] = 0;
   else
      for (k = i; k < j; k++) {
         t = MMM(C, d, i, k) + MMM(C, d, k + 1, j) + d[i - 1] * d[k] * d[j];
         if (t < C[i, j])
            C[i, j] = t;
      }
   return C[i, j];
}
```

Ví dụ: Bài toán túi xách 0/1

Hệ thức truy hồi của bài toán là:

$$T[i,j] = \begin{cases} max\{T[i-1,j], v_i + T[i-1, j-w_i]\} & j \ge w_i \\ T[i-1,j] & j < w_i \end{cases}$$

Điều kiện đầu được xác định như sau:

$$\begin{cases} T[0,j] = 0 & j \ge 0 \\ T[i,0] = 0 & i \ge 0 \end{cases}$$

Giải thuật

```
Knapsack(T[0 .. n, 0 .. W] , i, j) {
   if (T[i, j] < 0) {
      if (j < w[i])
         tmp = Knapsack(T, i - 1, j);
      else
         tmp = max(Knapsack(T, i - 1, j), v[i] + Knapsack(T, i-1, j – w[i]));
      T[i, j] = tmp;
   }
   return   T[i, j];
}
Knapsack_Memo() { // global variable: w[1 .. n], v[1 .. n]
   T[,] = -1;
   T[0, 0 .. W] = T[0 .. n, 0] = 0;
   return   Knapsack(T, n, W);
}
```

Nhận xét: Dữ liệu đầu vào là tập gồm 4 đồ vật: $\{w_1 = 2, v_1 = 12đ\}, \{w_2 = 1, v_2 = 10đ\}, \{w_3 = 3, v_3 = 20đ\}, \{w_4 = 2, v_4 = 15đ\}$ và $W = 5$:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	–1	12	22	–1	22
3	0	–1	–1	22	–1	32
4	0	–1	–1	–1	–1	**37**