TRƯỜNG ĐẠI HỌC ĐÀ LẠT

KHOA CÔNG NGHỆ THÔNG TIN

----------oOo----------

TOÁN RỜI RẠC

CHƯƠNG III: KHÁI NIỆM CĂN BẢN VỀ THUẬT TOÁN

I. Thuật toán

1. Định nghĩa: Thuật toán là một quy tắc để, với những dữ liệu ban đầu đã cho, tìm được lời giải của bài toán đang xét sau một khoảng thời gian hữu hạn.

Ví dụ 1:

Thuật toán tìm số lớn nhất trong 3 số thực a,b,c

- Input: a,b,c
- Output: x là số lớn nhất trong 3 số a,b,c
- Bước 1: Nếu a > b thì đặt x = a; ngược lại, đặt x = b.
- Bước 2: Nếu c > x thì đặt x = c.

I. Các thuật toán cơ bản

Ví dụ 2:

Thuật toán này tìm số lớn nhất trong dãy hữu hạn các số thực $s_1, s_2, \ldots, s_n$.

Vào: dãy hữu hạn các số thực $s_1, s_2, \ldots, s_n$.
Ra: $x = \max\{s_i \mid i = 1, 2, \ldots, n\}$.

Bước 1. Đặt $x := s_1$.

Bước 2. Với $i := 2$ đến n thực hiện Bước 3.

Bước 3. Nếu $s_i > x$ thì gán $x := s_i$.

I. Thuật toán

Một thuật toán phải có các tính chất chung như sau

Một *thuật toán* là một tập hợp các chỉ thị có những đặc trưng sau:

- *Tính chính xác.* Các bước được phát biểu một cách chính xác.
- *Tính duy nhất.* Các kết quả trung gian trong mỗi bước thực hiện được xác định một cách duy nhất và chỉ phụ thuộc vào dữ liệu đưa vào và các kết quả của bước trước.
- *Tính hữu hạn.* Thuật toán dừng sau hữu hạn bước.
- *Đầu vào.* Thuật toán có dữ liệu vào.
- *Đầu ra.* Thuật toán có dữ liệu ra.
- *Tính tổng quát.* Thuật toán thực hiện trên một tập các dữ liệu vào.

Chương VI: THUẬT TOÁN

- Thuật toán Euclid
- Thuật toán đệ quy
- Thuật toán tìm đường đi ngắn nhất trên đồ thị
- Độ phức tạp của thuật toán

II. Thuật toán Euclid

Ví dụ Các ước số nguyên dương của số 30 là

$$1, 2, 3, 5, 6, 10, 15, 30$$

và các ước số nguyên dương của số 105 là

$$1, 3, 5, 7, 15, 21, 35, 105;$$

do vậy các ước số chung dương của 30 và 105 là

$$1, 3, 5, 15.$$

Suy ra ước số chung lớn nhất của 30 và 105 là $\mathrm{USCLN}(30, 105) = 15$.

II. Thuật toán Euclid

Thuật toán Euclid

Thuật toán này tìm ước số chung lớn nhất của hai số tự nhiên a và b, trong đó a, b không đồng thời bằng 0.

Vào: a, b là số tự nhiên không đồng thời bằng 0.
Ra: USCLN là ước số chung lớn nhất của a và b.

Bước 1. Nếu $a < b$ thì hoán đổi a và b.

Bước 2. Nếu $b = 0$ thì thực hiện USCLN $:= a$ và dừng.

Bước 3. Chia a cho b và nhận được $a = bq + r$ với $0 \leq r < b$.

Bước 4. Thực hiện $a := b, b := r$ và chuyển đến Bước 2.

II. Thuật toán Euclid

Thuật toán Euclid

Bước 1. Nếu $a < b$ thì hoán đổi a và b.

Bước 2. Nếu $b = 0$ thì thực hiện USCLN := a và dừng.

Bước 3. Chia a cho b và nhận được $a = bq + r$ với $0 \leq r < b$.

Bước 4. Thực hiện $a := b, b := r$ và chuyển đến Bước 2.

Ví dụ Ta sẽ áp dụng thuật toán Euclid để tính USCLN(504, 396).

Đặt $a := 504, b := 396$. Vì $a > b$ nên ta chuyển đến Bước 2. Vì $b \neq 0$ nên chuyển đến Bước 3. Thực hiện Bước 3 ta có

$$504 = 396 \times 1 + 108.$$

Kế tiếp ta thực hiện Bước 4: đặt $a := 396, b := 108$ và chuyển đến Bước 2.

Vì $b \neq 0$ nên thực hiện Bước 3:

$$396 = 108 \times 3 + 72.$$

Thực hiện Bước 4: đặt $a := 108, b := 72$ và chuyển đến Bước 2.

II. Thuật toán Euclid

Thuật toán Euclid

Bước 1. Nếu $a < b$ thì hoán đổi a và b.

Bước 2. Nếu $b = 0$ thì thực hiện USCLN := a và dừng.

Bước 3. Chia a cho b và nhận được $a = bq + r$ với $0 \leq r < b$.

Bước 4. Thực hiện $a := b, b := r$ và chuyển đến Bước 2.

Vì $b \neq 0$ nên thực hiện Bước 3:

$$108 = 72 \times 1 + 36.$$

Thực hiện Bước 4: đặt $a := 72, b := 36$ và chuyển đến Bước 2.

Vì $b \neq 0$ nên thực hiện Bước 3:

$$72 = 36 \times 2 + 0.$$

Thực hiện Bước 4: đặt $a := 36, b := 0$ và chuyển đến Bước 2.

Vì $b = 0$ nên áp dụng Bước 2 có USCLN $:= a = 36$ và thuật toán dừng. USCLN$(504, 396) = 36$.

Thuật toán đệ quy

Thuật toán đệ quy là một thuật toán gọi lại chính nó. Đệ quy là một công cụ hữu dụng và tự nhiên để giải quyết một lớp lớn các bài toán. Để giải những bài toán trong lớp này ta có thể sử dụng kỹ thuật *chia để trị*: Bài toán cần giải quyết được chia thành những bài toán con có dạng như bài toán ban đầu. Mỗi bài toán con lại được phân rã thêm. Quá trình phân rã cho đến khi nhận được những bài toán con với lời giải dễ dàng. Cuối cùng, tổ hợp các lời giải của các bài toán con ta được lời giải của bài toán ban đầu.

Ví dụ 1. n giai thừa của số tự nhiên n là số nguyên dương xác định bởi

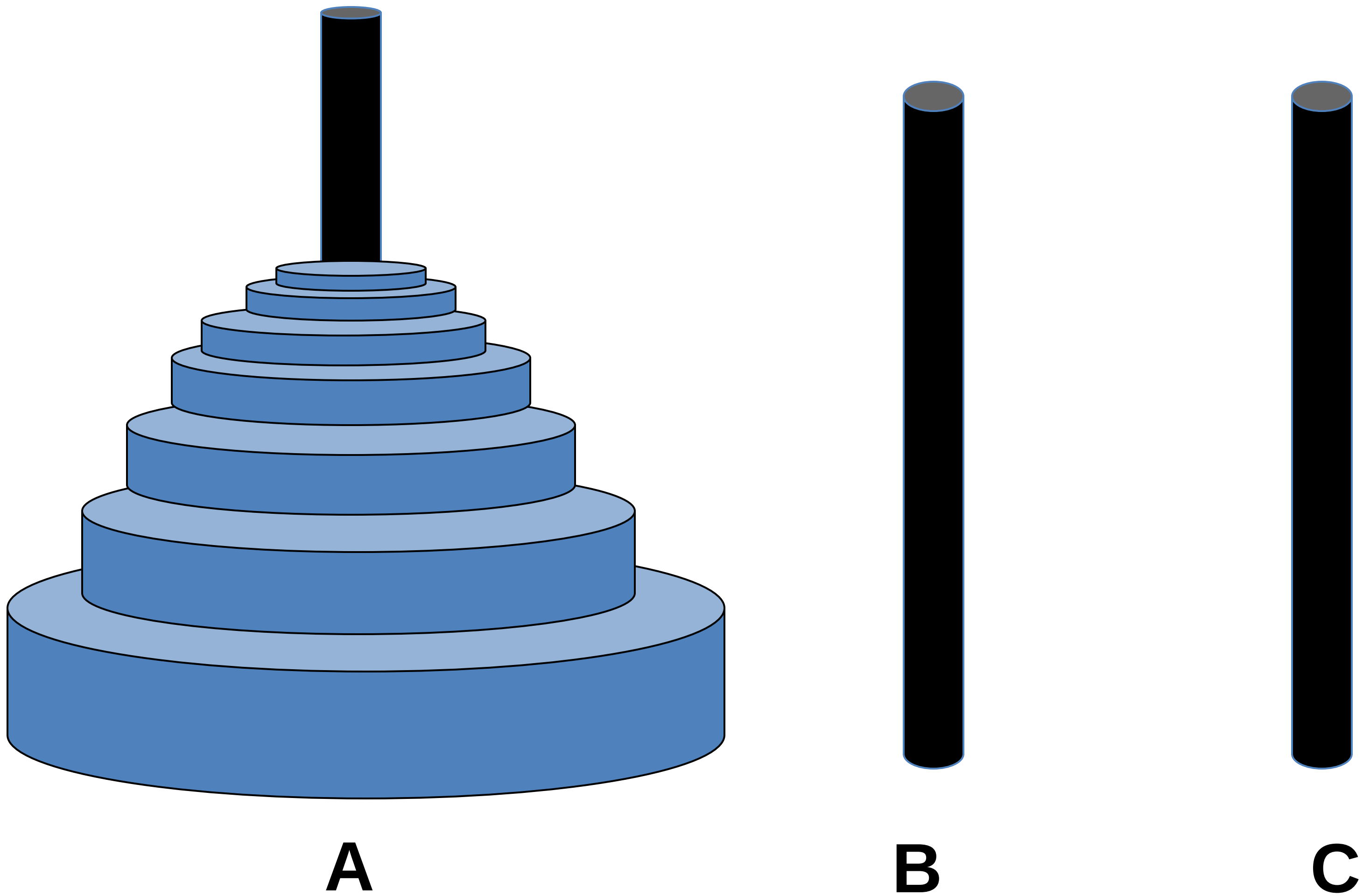
$$n! := \begin{cases} 1 & \text{nếu } n = 0, \\ n(n-1)(n-2)\cdots 2\cdot 1 & \text{nếu } n \geq 1. \end{cases}$$

Tức là nếu $n \geq 1$ thì $n!$ bằng tích của tất cả các số tự nhiên từ 1 đến n. Chẳng hạn,

$$\begin{aligned} 3! &= 3\cdot 2\cdot 1 = 6, \\ 6! &= 6\cdot 5\cdot 4\cdot 3\cdot 2\cdot 1 = 720. \end{aligned}$$

IV. Hệ thức đệ qui

Ví dụ 2. Tháp Hà Nội



IV. Hệ thức đệ qui

Có 3 cọc A, B, C và n đĩa (có lỗ để đặt vào cọc) với đường kính đôi một khác nhau. Nguyên tắc đặt đĩa vào cọc là: mỗi đĩa chỉ được chồng lên đĩa lớn hơn nó. Ban đầu, cả n đĩa được đặt chồng lên nhau ở cọc A, hai cọc B và C để trống. Vấn đề đặt ra là chuyển cả n đĩa ở cọc A sang cọc C (có thể qua trung gian cọc B), mỗi lần chỉ chuyển một đĩa. Gọi x_n *là số lần chuyển đĩa. Tìm một hệ thức đệ qui cho* x_n

IV. Hệ thức đệ qui

Giải.

- Với n = 1 ta có $x_1 = 1$.

- Với n > 1, trước hết ta chuyển n-1 đĩa bên trên sang cọc B qua trung gian cọc C (giữ nguyên đĩa thứ n dưới cùng ở cọc A). Số lần chuyển n-1 đĩa đó là x_{n-1}. Sau đó ta chuyển đĩa thứ n từ cọc A sang cọc C. Cuối cùng ta chuyển n-1 đĩa từ cọc B sang cọc C. Số lần chuyển n-1 đĩa đó lại là x_{n-1}.

IV. Hệ thức đệ qui

Như vậy số lần chuyển tòan bộ n đĩa từ A sang C là:

$x_{n-1} + 1 + x_{n-1} = 2x_{n-1} + 1.$

Nghĩa là $x_n = 2x_{n-1} + 1$, ta có hệ thức đệ qui tuyến tính không thuần nhất cấp 1:

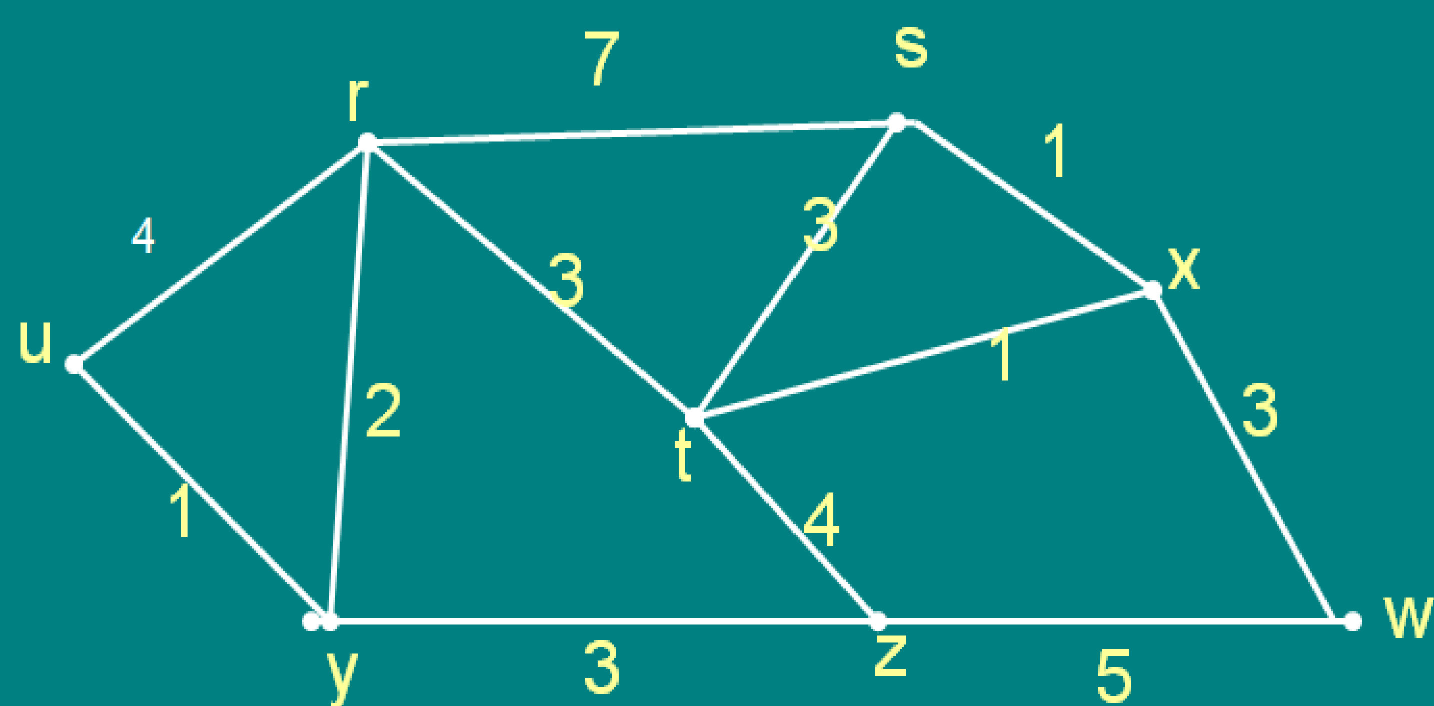
$$\begin{cases} x_n = 2x_{n-1} + 1; \\ x_1 = 1. \end{cases}$$

IV. Hệ thức đệ qui

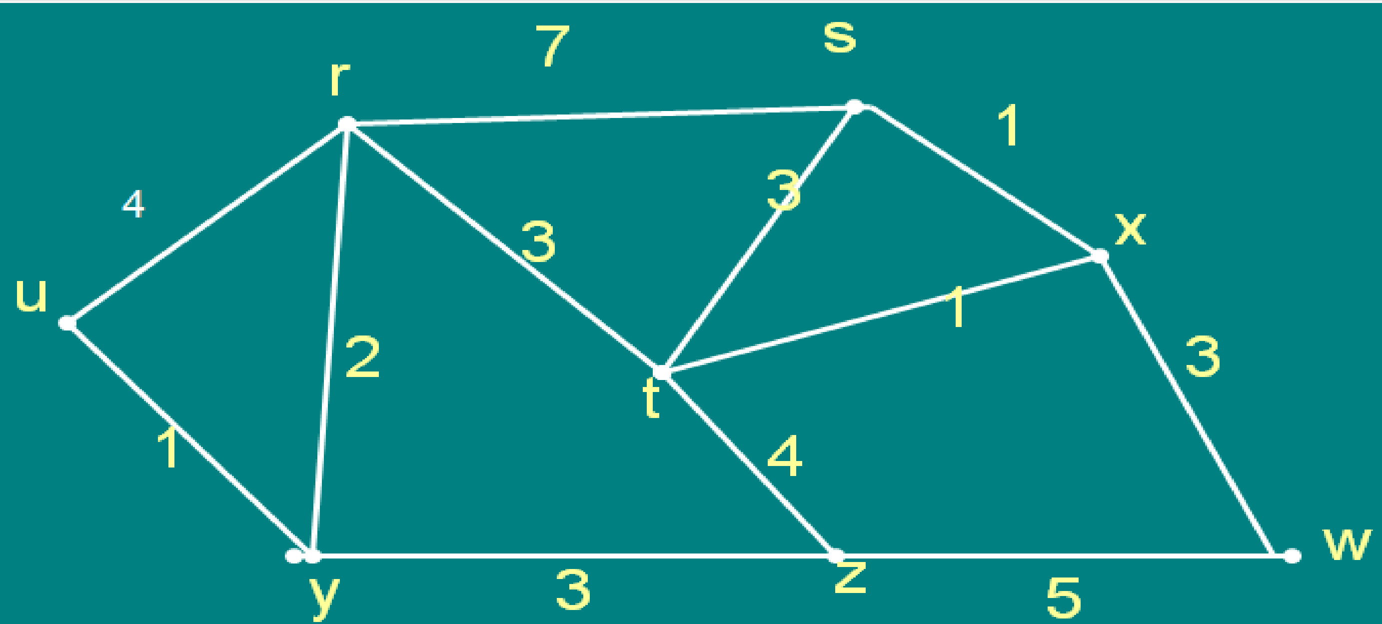
```
private void ThapHN(ref int soBuoc, int Sodia, string from, string inter, string to)
    {
        if (Sodia == 1) {
            listBox_FieldList.Items.Add("Đĩa 1 từ " + from + " to " + to);
            soBuoc = soBuoc + 1;
            //MessageBox.Show("Đĩa 1 từ " + from + " to " + to);
        }
        else
        {
            ThapHN(ref soBuoc, Sodia - 1, from, to, inter); // chuyển n-1 đĩa sang B qua
trung gian C
         //listBox_FieldList.Items.Add("Đĩa " + Sodia.ToString() + " từ " + from + " to " + to);
            soBuoc = soBuoc + 1;
            ThapHN(ref soBuoc, Sodia - 1, inter, from, to);// chuyển n-1 đĩa từ cọc B sang
cọc C.
        }
    }
```

Thuật toán Dijkstra

Bài tập 1. Tìm đường đi ngắn nhất từ u_0 đến các đỉnh còn lại

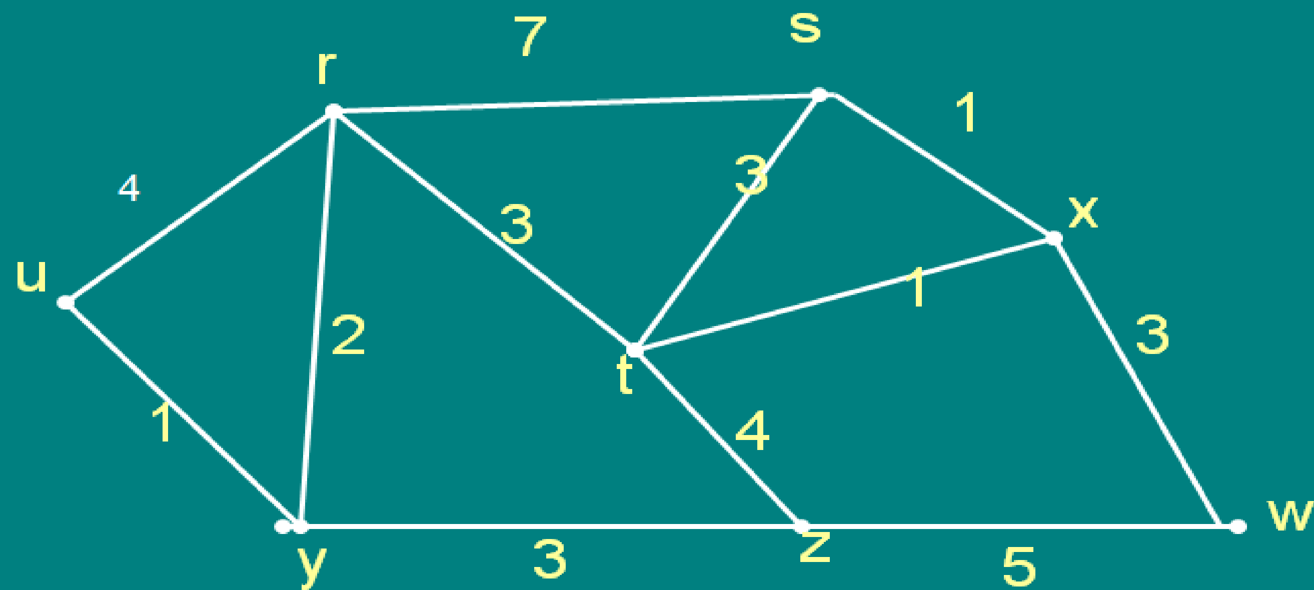


Thuật toán Dijkstra



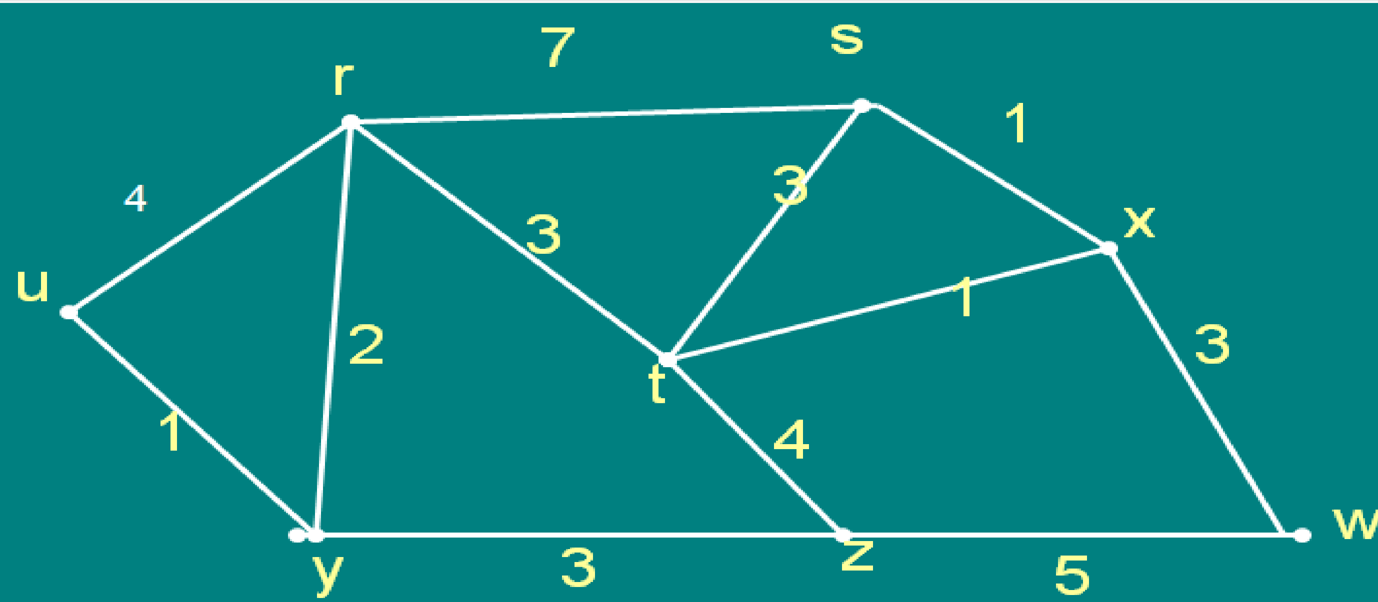
u_0	r	s	t	x	y	z	w
0*	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)

Thuật toán Dijkstra



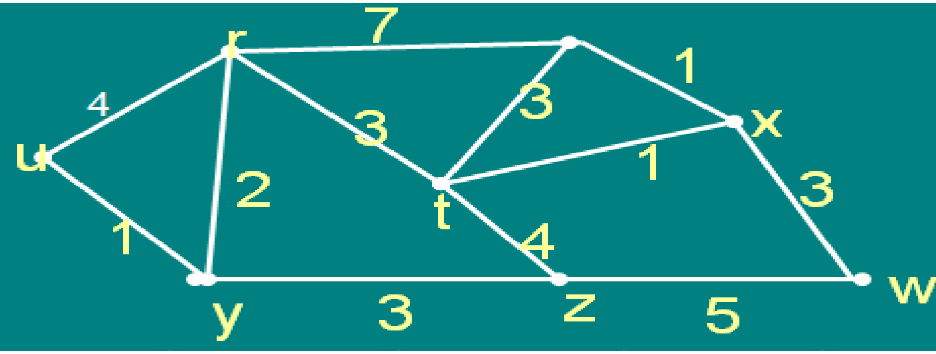
u_0	r	s	t	x	y	z	w
0*	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)	(∞,–)
–	$(4,u_0)$	(∞,–)	(∞,–)	(∞,–)	$(1u_0)^*$	(∞,–)	(∞,–)

Thuật toán Dijkstra



u_0	r	s	t	x	y	z	w
0*	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)
-	$(4,u_0)$	(∞,-)	(∞,-)	(∞,-)	$(1u_0)$*	(∞,-)	(∞,-)
-	(3,y)*	(∞,-)	(∞,-)	(∞,-)	-	(4,y)	(∞,-)

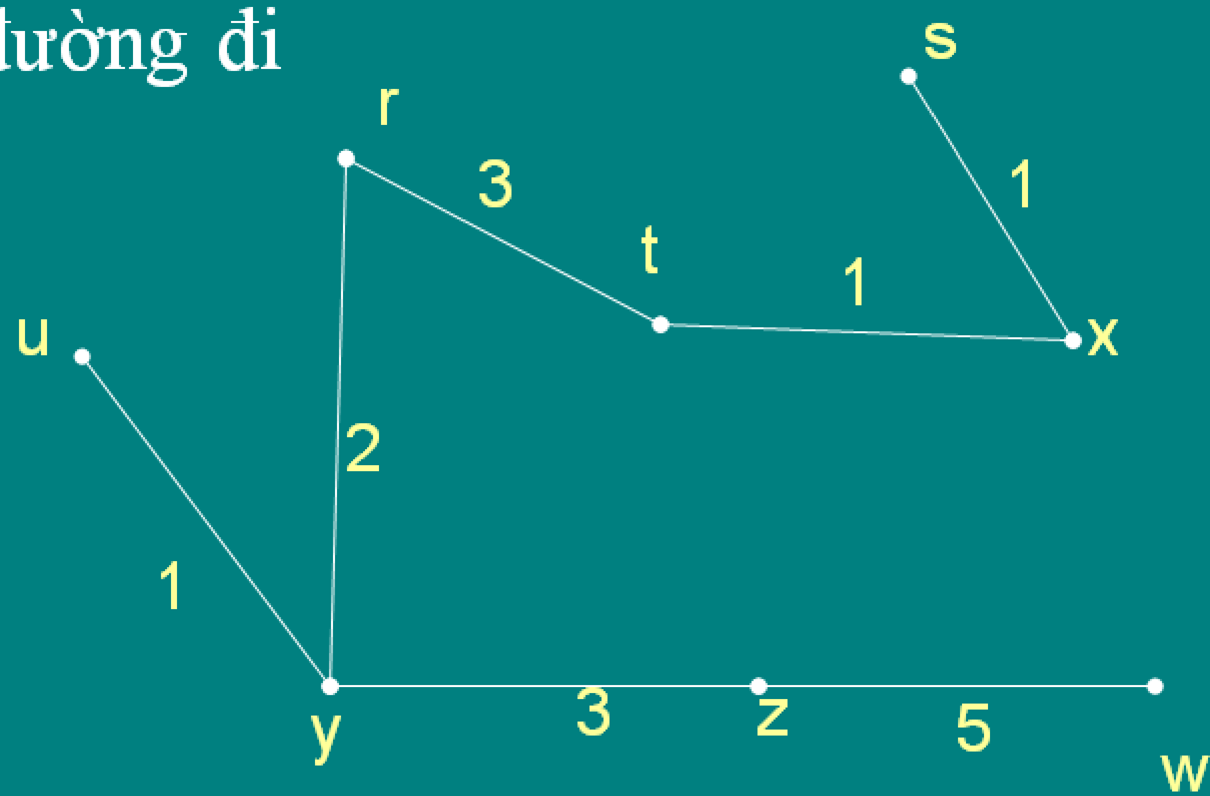
Thuật toán Dijkstra



u_0	r	s	t	x	y	z	w
0*	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)
-	$(4,u_0)$	(∞,-)	(∞,-)	(∞,-)	$(1u_0)^*$	(∞,-)	(∞,-)
-	(3,y)*	(∞,-)	(∞,-)	(∞,-)	-	(4,y)	(∞,-)
-	-	(10,r)	(6,r)	(∞,-)	-	(4,y)*	(∞,-)
-	-	(10,r)	(6,r)*	(∞,-)	-	-	(9,z)
-	-	(9,t)	-	(7,t)*	-	-	(9,z)
-	-	(8,x)*	-	-	-	-	(9,z)
-	-	-	-	-	-	-	106 (9,z)*

Thuật toán Dijkstra

Cây đường đi



Thuật toán Dijkstra

Bước1. $i:=0$, $S:=V\backslash\{u_0\}$, $L(u_0):=0$, $L(v):=\infty$ với mọi $v \in S$ và đánh dấu đỉnh v bởi$(\infty,-)$. Nếu $n=1$ thì xuất $d(u_0,u_0)=0=L(u_0)$

Bước2. Với mọi $v \in S$ và kề với u_i(nếu đồ thị có hướng thì v là đỉnh sau của u_i), đặt $L(v):= \min\{L(v),L(u_i)+w(u_i\ v)\}$.Xác định $k = \min L(v)$,$v\in S$.

Nếu $k= L(v_j)$ thì xuất $d(u_0,v_j)= k$ và đánh dấu v_j bởi $(L(v_j);u_i)$.

$u_{i+1}:= v_j$ $S:=S\backslash\{u_{i+1}\}$

Bước3 $i:=i+1$

Nếu $i = n-1$ thì kết thúc

Nếu không thì quay lại Bước 2

Độ phức tạp của thuật toán

Một bài toán có nhiều cách giải. Để so sánh các cách giải với nhau, ta có thể dựa vào ***độ phức tạp của thuật toán***. Độ phức tạp của thuật toán có thể chia làm hai loại:

• **Độ phức tạp không gian**: Số ô nhớ cần thiết để có thể triển khai thực hiện thuật toán.

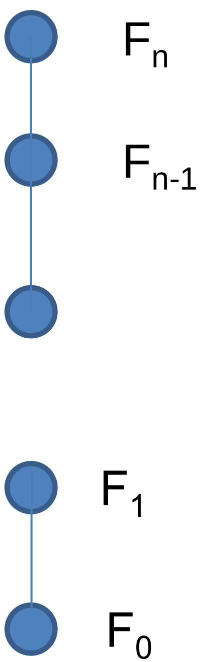
• **Độ phức tạp thời gian**: Thời gian cần thiết để thực hiện xong các bước các bước của thuật toán, cho ra kết quả (sử dụng hết RAM để thực hiện thuật toán).

III. Độ phức tạp của thuật toán

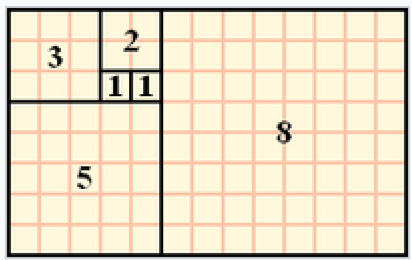
Độ phức tạp không gian

Ví dụ. Tính n! hoặc dãy Fibonacci

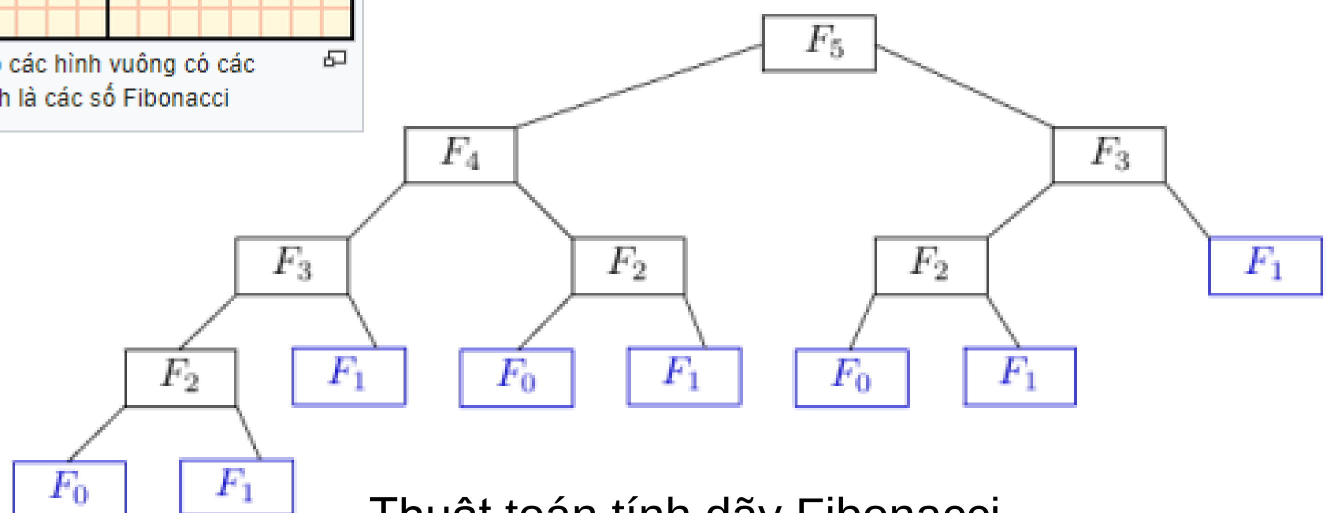
$$F(n) := \begin{cases} 1, & \text{khi } n = 1; \\ 1, & \text{khi } n = 2; \\ F(n-1) + F(n-2) & \text{khi } n > 2. \end{cases}$$



Thuật toán tính n!



Xếp các hình vuông có các cạnh là các số Fibonacci



Thuật toán tính dãy Fibonacci

=> Ví dụ tràn không gia bộ nhớ

III. Độ phức tạp của thuật toán

Độ phức tạp thời gian

(a) *Thời gian trường hợp tốt nhất* là thời gian ít nhất để thực hiện thuật toán.

(b) *Thời gian trường hợp xấu nhất* là thời gian nhiều nhất để thực hiện thuật toán.

(c) *Thời gian trường hợp trung bình* là thời gian trung bình để thực hiện thuật toán.

Ví dụ Xét thuật toán tìm kiếm trong một dãy không được sắp thứ tự sau:

Vào: $s, s_1, s_2, \ldots, s_n$.
Ra: $j = 0$ nếu $s \neq s_i$ với mọi i; ngược lại, j là chỉ số nhỏ nhất sao cho $s = s_j$.

Bước 1. Với $i := 1$ đến n thực hiện Bước 2.

Bước 2. Nếu $s = s_i$ thì đặt $j := i$ và dừng thuật toán.

Bước 3. Đặt $j := 0$.

Có thể chứng minh thời gian trường hợp tốt nhất bằng $O(1)$, thời gian trường hợp xấu nhất bằng thời gian trường hợp trung bình và bằng $O(n)$.

Độ phức tạp của thuật toán

Độ phức tạp của thuật toán

Size (n)	Constant(C=1)	log2n	n	nlog2n	n^2	n^3	2^n	n!
1		0	1	0	1	1	2	1
2		1	2	2	4	8	4	2
4		2	4	8	16	64	16	24
8		3	8	24	64	512	256	40320
16		4	16	64	256	4096	65536	2.09228E+13
32		5	32	160	1024	32768	4294967296	2.63131E+35
64		6	64	384	4096	262144	1.8447E+19	1.26887E+89
	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$	$O(n!)$
	Hằng số	Logarit	Tuyến tính	$n \log n$	Bình phương	Lập phương	Mũ	Giai thừa

F(n) is running time

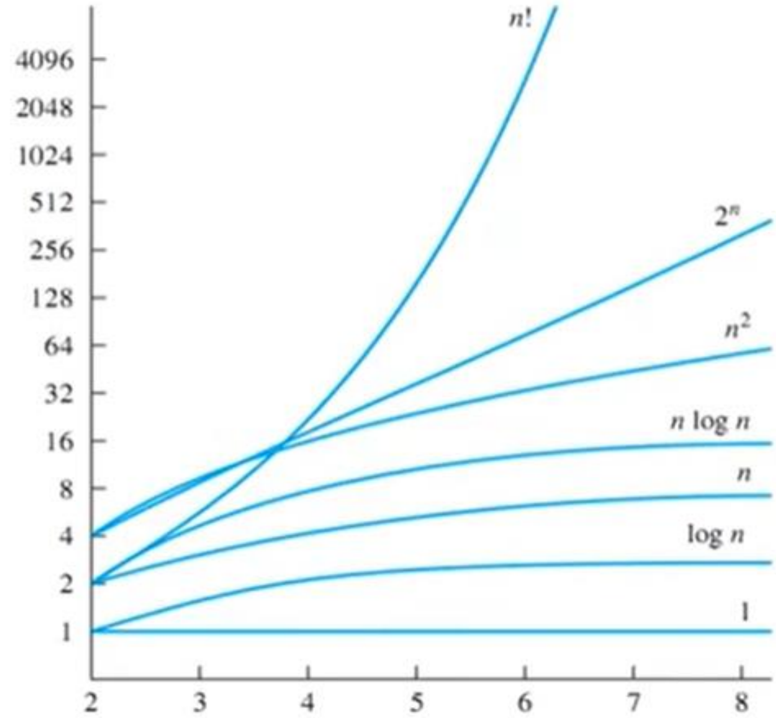
So sánh độ tăng của các hàm, Constant không thay đổi

The growth of important functions

$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n!$

The growth of n <= growth of $n\log_2 n$

The growth of n^2 <= growth of 2^n



ie:

➔ n is $O(n\log_2 n)$

➔ n^2 is $O(n^2)$

? $3*n^2$ is ?

?3*n^2	n	n^2	3*n^2
	1	1	3
	2	4	12
	3	9	27

3*n^2

--> 3*n^2 is O(n^2)

n^2

Độ phức tạp của thuật toán

- Khi xét *Big-O*, ta luôn xét với chiều dài dữ liệu đầu vào *n* là 1 số vô cùng lớn, do đó có thể coi *n* là $+\infty$.
- Các bước thực thi không phụ thuộc vào giá trị đầu vào(ví dụ: phép tính toán, gán, so sánh,...) có độ phức tạp hằng số(**O(1)**).
- Tổng các hằng số là 1 **hằng số**.
- Tích của 1 số dương với $+\infty$ cũng là $+\infty$**,** tổng của 1 số với $+\infty$ cũng là$+\infty$.

How to estimate big-O of other functions?

- How to estimate big-oh of functions such as

$100n^2 + 23n\log(n^3) + 2017$?

How to estimate big-O of other functions?

- How to estimate big-oh of functions such as $100n^2 + 23n\log(n^3) + 2017$? is $O(n^2)$

$1 \cdot n^2 + 1.\cancel{n\log n} + \cancel{1}$

n^2 ✓

STT	CÔNG THỨC LOGARIT
1	$\log_a 1 = 0$
2	$\log_a a = 1$
3	$\log_a a^M = M$
4	$a^{\log_a N} = N$
5	$\log_a(N_1.N_2) = \log_a N_1 + \log_a N_2$
6	$\log_a(\frac{N_1}{N_2}) = \log_a N_1 - \log_a N_2$
7	$\log_a N^\alpha = \alpha.\log_a N$
8	$\log_a N^2 = 2.\log_a \lvert N \rvert$
9	$\log_a N = \log_a b.\log_b N$
10	$\log_b N = \frac{\log_a N}{\log_a b}$
11	$\log_a b = \frac{1}{\log_b a}$
12	$\log_{a^\alpha} N = \frac{1}{\alpha}\log_a N$
13	$a^{\log_b c} = c^{\log_b a}$

Độ phức tạp của thuật toán

Estimate big-oh

```
procedure printsth(n: positive integer)
begin
    for i:=1 to n do
            print "hi"
    for k:=1 to n do
            print "I love you"
end
```

$n + n \rightarrow O(n)$

Estimate big-O of the given algorithm

Độ phức tạp của thuật toán

Estimate big-oh

```
procedure printsth(n: positive integer)
begin
    for i:=1 to n do
        for k:=1 to n do
            print "I love you"
end
```

$n^2 \rightarrow O(n^2)$

Estimate big-O of the given algorithm

Độ phức tạp của thuật toán

ALGORITHM 4 The Bubble Sort.

procedure *bubblesort*$(a_1, \ldots, a_n:$ real numbers with $n \geq 2)$
for $i := 1$ **to** $n - 1$
  for $j := 1$ **to** $n - i$
    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, \ldots, a_n$ is in increasing order$\}$

➔ $O(n^2)$ time complexity

Độ phức tạp của thuật toán

Estimate big-oh

```
procedure printsth(n: positive integer)
begin
    for i:=1 to n do
        for k:=1 to n do
            for j:=1 to n print do
                print "I love you"
end
```

$O(n^3)$

Estimate big-O of the given algorithm

Độ phức tạp của thuật toán

ALGORITHM 1 Matrix Multiplication.

procedure *matrix multiplication*(**A**, **B**: matrices)
for $i := 1$ **to** m
    for $j := 1$ **to** n
        $c_{ij} := 0$
        for $q := 1$ **to** k
            $c_{ij} := c_{ij} + a_{iq}b_{qj}$
return C $\{\mathbf{C} = [c_{ij}]$ is the product of **A** and **B**$\}$

→ $O(n^3)$ time complexity

Nhân 2 ma trận, đồ họa, Game

Độ phức tạp của thuật toán

Estimate big-oh

```
procedure printsth(n: positive integer)
begin
      for i:=1 to n do
            print "hi"
      for j:=1 to n do
            for k:=1 to i do
                  print "I love you"
end
```

$n + n^2 \rightarrow O(n^2)$

Estimate big-O of the given algorithm

Thuật toán 1

Code	Cost
`int count_1( int N)`	
`{`	
`  sum = 0`	1
`  for (i=1; i<=n; i++) {`	$2N$
`    for (j=i; j<=n; j++) {`	$2\sum_{i=1}^{N}(N+1-i)$
`      sum++`	$\sum_{i=1}^{N}(N+1-i)$
`    }`	
`  }`	
`  return sum`	1
`}`	

Thời gian chạy: $2+2N+3\sum_{i=1}^{N}(N+1-i)=\frac{3}{2}N^2+\frac{7}{2}N+2$

Độ phức tạp của thuật toán

Thuật toán 1

Xét n = 5

i=1,2,3,4,5

i=1;

j=1,2,3,4,5 → j chạy 5 lần

i=2;

j=2,3,4,5 → j chạy 4 lần

i=3;

j=3,4,5 → j chạy 3 lần

i=4

j=4,5 → j chạy 2 lần

i=5

j=5 → j chạy 1 lần

Tổng số lần j sẽ chạy: 5 + 4 + 3 + 2 + 1 lần = 1 + 2 + 3 + 4 + 5

Tổng quát nhập vào n

j sẽ chạy 1 + 2 + 3 + 4 + 5…+n-1 + n = n*(n+1)/2

Code	Cost
`int count_1( int N)`	
`{`	
`sum = 0`	1
`for (i=1; i<=n; i++) {`	$2N$
`for (j=i; j<=n; j++) {`	$2\sum_{i=1}^{N}(N+1-i)$
`sum++`	$\sum_{i=1}^{N}(N+1-i)$
`}`	
`}`	
`return sum`	1
`}`	

Thời gian chạy: $2+2N+3\sum_{i=1}^{N}(N+1-i)=\frac{3}{2}N^2+\frac{7}{2}N+2$

Độ phức tạp của thuật toán

$$\sum_{i=1}^{N}(N+1-i)=\sum_{i=1}^{N} i = N(N+1)/2$$

Thuật toán 1

```
1  int count_1( int N)
2  {
3    sum = 0                      1
4    for (i=1; i<=n; i++) {       2N
5      for (j=i; j<=n; j++) {     2∑_{i=1}^{N}(N+1−i)
6        sum++                    ∑_{i=1}^{N}(N+1−i)
7      }
8    }
9    return sum                   1
10 }
```

Thời gian chạy: $2+2N+3\sum_{i=1}^{N}(N+1-i)=\frac{3}{2}N^2+\frac{7}{2}N+2$

$1 + 2N + 3\sum_{i=1}^{N}(N + 1 - i) + 1 =$

$2 + 2N + 3\sum_{i=1}^{N}(N + 1 - i)$

2 + 2N + 3N(N+1)/2

2 + 2N + 3/2*N^2 +3/2*N

3/2*N^2 + 7/2*N + 2

Độ phức tạp của thuật toán

Thuật toán 2

```
int count_2( int N)
{
  sum = 0                          1
for (i=1; i<=n; i++){              2N
    sum += N+1-i                   3N
  }
  return sum                       1
}
```

Thời gian chạy: $5N+2$

$$\sum_{i=1}^{N}(N+1-i)=\sum_{i=1}^{N}i=N(N+1)/2$$

Độ phức tạp của thuật toán

Thuật toán 3

$$\sum_{i=1}^{N}(N+1-i)=\sum_{i=1}^{N}i=N(N+1)/2$$

```
int count_3( int N)
{
  sum = N(N+1)/2              ———————— 4
  return sum          ———————————————— 1
}
```

Thời gian chạy là 5 đơn vị thời gian