CHƯƠNG 1: KỸ THUẬT PHÂN TÍCH ĐÁNH GIÁ THUẬT TOÁN

- Định nghĩa độ phức tạp của thuật toán? So sánh các ***hàm tính độ phức tạp*** của thuật toán

- Tính thời gian thực hiện cho các đoạn chương trình không chứa đệ quy (dùng ***quy tắc cộng, quy tắc nhân***)

- Thành lập phương trình đệ quy và giải bằng ***phương pháp truy hồi*** hoặc ***phương pháp lời giải tổng quát***

I. Thuật toán

1. Định nghĩa

- Cách thức giải quyết vấn đề

2. Đặc tả thuật toán

- Không hình thức: Ngôn ngữ tự nhiên (Không phổ biến)

- Nửa hình thức: Lưu đồ (Flow - Chart), Sơ đồ khối

- Hình thức (Không phổ biến)

3. Sự cần thiết của đánh giá, thiết kế thuật toán

→ Chọn ra thuật toán tốt nhất

*** Tiêu chuẩn đánh giá:**

- Tính đúng đắn: + Chạy trên dữ liệu thử: Không khả thi

+ Chứng minh lý thuyết (bằng toán học): Khó khăn

- Tính đơn giản

- Tính nhanh chóng (thời gian thực thi)

+ Rất quan trọng khi chương trình thực thi nhiều lần

= Hiệu quả thời gian thực thi

II. Thời gian thực hiện chương trình

- Là một hàm của kích thước dữ liệu vào, ký hiệu **T(n),** trong đó n là *kích thước (độ lớn) của dữ liệu vào*

Ví dụ: Chương trình tính **tổng n số** có thời gian thực hiện là **T(n) = Cn** trong đó C là một hằng số

- Thời gian thực hiện chương trình là một *hàm không âm*, tức là $\mathbf{T(n) \geq 0, \forall n \geq 0.}$

- T(n) được tính trong trường hợp xấu nhất

- Đơn vị của T(n): xác định bởi ***số các lệnh/chỉ thị*** được thực hiện trong một *máy tính lý tưởng*

- Thời gian thực hiện **phụ thuộc** vào

+ Kích thước

+ Tính chất dữ liệu nhập vào

III. Tỷ suất tăng của hàm và độ phức tạp của thuật toán

1. Tỷ suất tăng của hàm

- Ta nói hàm không âm **T(n)** có **tỷ suất tăng** (*growth rate*) f(n) nếu tồn tại các hằng số C và N_0 sao cho $\mathbf{T(n) \leq C.f(n), \forall n \geq N_0}$

⇒ Tỷ suất tăng f(n) = tốc độ tăng của hàm khi n tăng

- Quy tắc tính tỷ suất tăng của hàm: Nếu T(n) là một đa thức của n thì tỷ suất tăng của T(n) là n với **số mũ cao nhất**

2. <u>Độ phức tạp của thuật toán</u>

- Tỷ suất tăng của hàm thời gian = **Độ phức tạp của thuật toán**

- Ký pháp Ô lớn (big-O notation): Cho một thuật toán P có *thời gian thực hiện* là hàm T(n), nếu T(n) có tỷ suất tăng là f(n) thì thuật toán P có độ phức tạp là f(n) và **ký hiệu thời gian thực hiện T(n) là O(f(n))**

- Tính chất: (1) O(C.f(n)) = O(f(n)) với C là hằng số.

(2) O(C) = O(1)

- Lưu ý: + Độ phức tạp của thuật toán là hàm chặn trên của hàm thời gian.

+ Hằng nhân tử C trong hàm chặn trên thường không có ý nghĩa.

- Ví dụ: + Thời gian thực hiện $T(n) = 3n^3 + 2n^2$

+ Tỷ suất tăng $f(n) = n^2$

+ Độ phức tạp $O(f(n)) = O(n^2)$

- Các hàm độ phức tạp thường gặp:

Dạng O	Tên phân loại	
O(1)	Hằng	~ logn
O($\log_2 n$)	Logarit	
O($\sqrt{n}$)	Căn thức	
O($\sqrt[3]{n}$)		
…		
O($\sqrt[m]{n}$)		
O(n)	Tuyến tính	Đa thức (có thể chấp nhận được)
O(n^2)	Bình phương	
O(n^3)	Bậc ba	
…		
O(n^m)	Đa thức	
O(c^n), với c > 1	Mũ	Độ phức tạp lớn (cần phải cải tiến)
O(n!)	Giai thừa	

- Thứ tự độ phức tạp tăng dần:
  + Hàm ***hằng***: O(1)
  + Hàm ***logarit***: O(logn)
  + Hàm ***tuyến tính***: O(n)
  + Hàm ***logarit tuyến tính***: O(nlogn)
  + Hàm ***đa thức***: O(n^c)
  + Hàm ***mũ***: O(C^n)
  + Hàm ***giai thừa***: O(n!)

*** Bài tập kiểm tra:** phân loại và sắp xếp từ thấp đến cao

IV. Cách tính độ phức tạp

- Cho 2 đoạn chương trình:
  + P1 có thời gian thực hiện T1(n) = O(f1(n))
  + P2 có thời gian thực hiện T2(n) = O(f2(n))

* Quy tắc cộng: Thời gian thực hiện P1 và P2 **nối tiếp** nhau sẽ là:
T(n) = T1(n) + T2(n) = O(max(f1(n), f2(n)))

Kí hiệu:

(+) ↳ **P_1;**
P_2;

* Quy tắc nhân: Thời gian thực hiện P1 và P2 **lồng nhau** (chẳng hạn vòng lặp lồng nhau) sẽ là:

T(n) = T1(n) x T2(n) = O(f1(n) x f2(n))

Kí hiệu

(x) **P_1;**

P_2;

*** Quy ước thời gian của các lệnh:**

- Lệnh đọc **(read, scanf),** lệnh ghi **(write, printf),** lệnh **gán,** lệnh **return, định trị biểu thức, so sánh:** Thời gian = hằng số hay **O(1)**

- Lệnh if: if(điều kiện) lệnh 1; else lệnh 2; ⟹ Thời gian = max(lệnh 1, lệnh 2) + điều kiện = **max(lệnh 1, lệnh 2, điều kiện)**

- Vòng lặp:

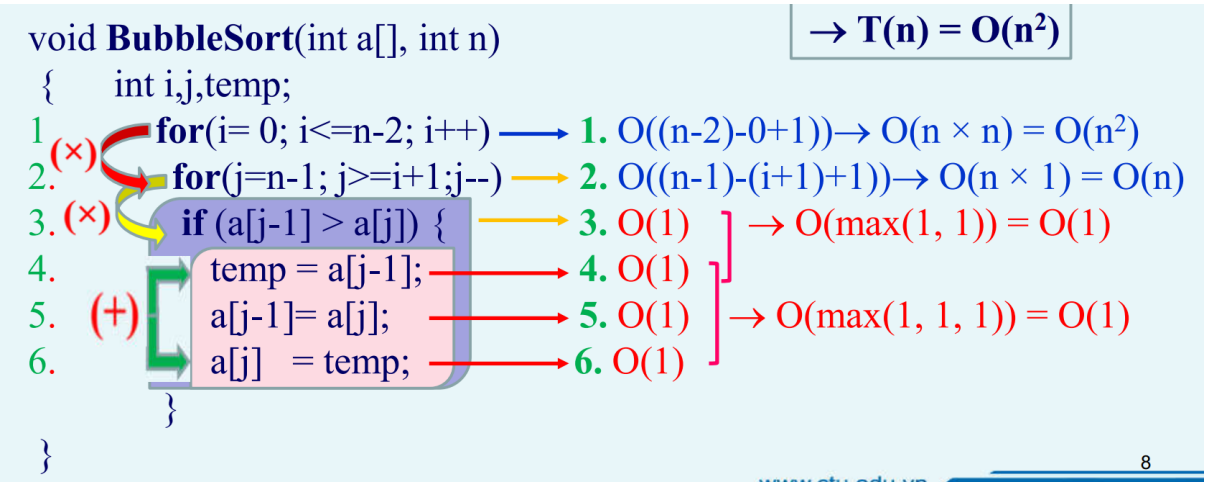
+ Thời gian = tổng thời gian thực hiện thân vòng lặp

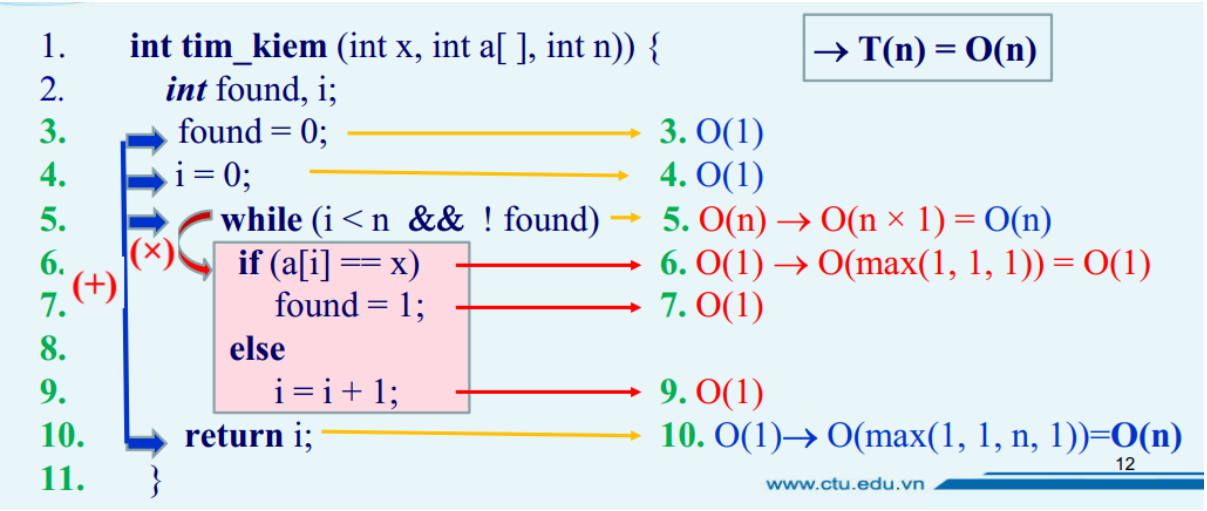
+ Vòng lặp có số lần lặp xác định (**for**): tính số lần lặp

+ Vòng lặp có số lần lặp không xác định (**while**): tính số lần lặp trong *trường hợp xấu nhất* (lặp nhiều nhất)

* Xét phương pháp tính độ phức tạp trong 3 trường hợp:

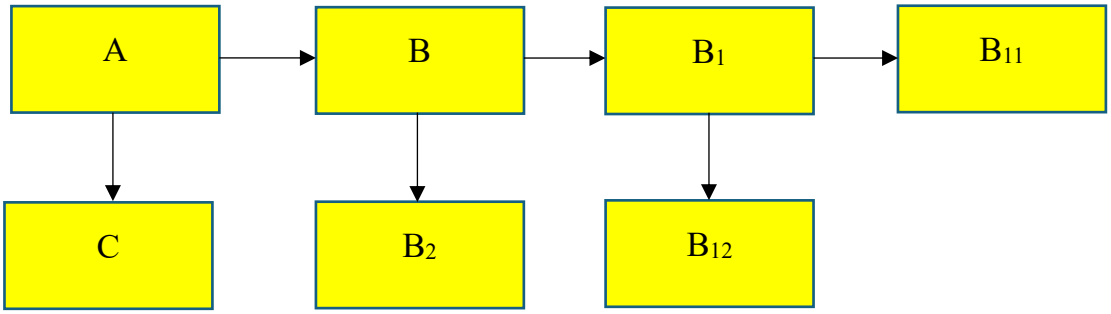
(1) Chương trình không gọi chương trình con



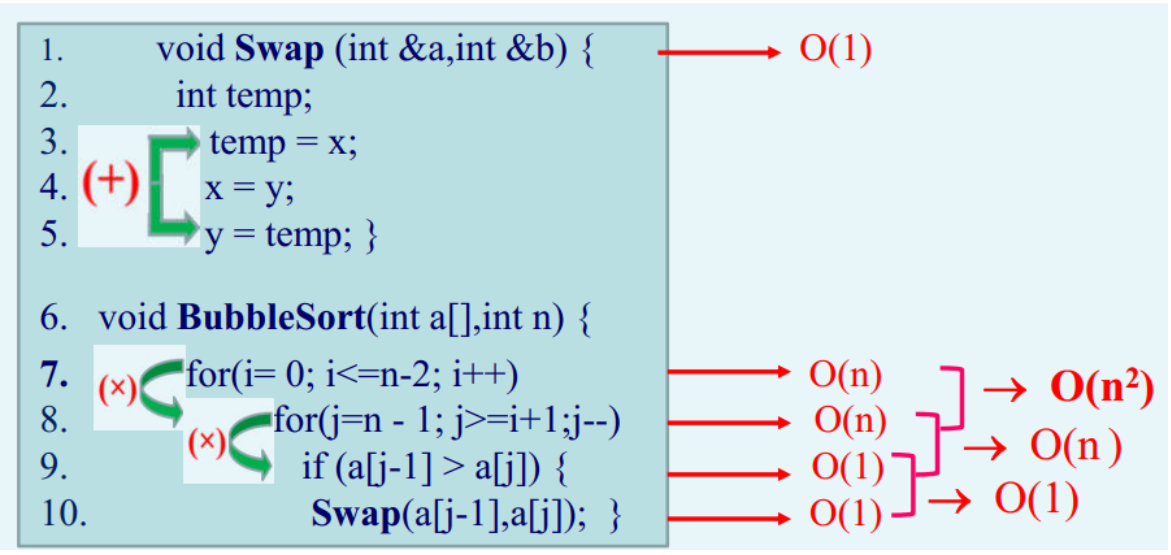


(2) Chương trình có gọi chương trình con *không đệ quy*

*** Quy tắc tính từ trong ra ngoài:**



C → B2 → B12 → B11 → B1→ B → A



(3) Chương trình đệ quy

* 2 dạng chương trình đệ quy:

- Đệ quy trực tiếp - Đệ quy gián tiếp

*** Tính độ phức tạp chương trình đệ quy:**

(1) Thành lập phương trình đệ quy T(n)

- Phương trình đệ quy là phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) và T(k) là thời gian thực hiện chương trình có kích thước dữ liệu nhập tương ứng là n và k, với k < n.

- Để thành lập chương trình đệ quy, phải căn cứ vào chương trình đệ quy.

+ Khi đệ quy dừng: xem xét khi đó **chương trình làm gì và tốn hết bao nhiêu thời gian** (C(n))

+ Khi đệ quy chưa dừng: xem xét có **bao nhiêu lời gọi đệ quy với kích thước k** thì sẽ có bao nhiêu **T(k).**

- Ngoài ra, còn phải xem xét thời gian phân chia bài toán và tổng hợp các lời giải (d(n))

* Dạng tổng quát của một **phương trình đệ quy** là:

$$T(n) = \begin{cases} C(n) & \leftarrow \text{Khi đệ quy dừng} \\ F(T(k)) + d(n) & \leftarrow \text{Khi đệ quy chưa dừng} \end{cases}$$

- Mối quan hệ giữa T(n) và T(k): bao nhiêu lời gọi

Ví dụ 1: Hàm tính giai thừa

- Gọi T(n) là thời gian tính n!

- Thì T(n-1) là thời gian tính (n-1)!

```
1  int giaithua(int n){
2      if(n == 0) return 1;
3      else return n * giaithua(n-1);
4  }
```

- Trong trường hợp n = 0: thực hiện return 1 tốn O(1) → T(0) = C_1

- Trong trường hợp n > 0: gọi đệ quy giaithua(n-1), tốn T(n - 1)

→ T(n) = T(n-1)

- Sau khi có kết quả từ việc gọi đệ quy, chương trình phải nhân kết quả đó với n và trả về tích số. Thời gian thực hiện phép nhân và trả kết quả về là một hằng C_2

- Vậy ta có phương trình đệ quy như sau

$$T(n) = \begin{cases} C_1 & nếu\ n = 0 \\ T(n-1) + C_2 & nếu\ n > 0 \end{cases}$$ O(n)

VD2: Hàm **mergeSort**

```
// Hàm merge sort sử dụng đệ quy
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}
```

- T(n): mergeSort (n phần tử)

- T(n/2): mergeSort (n/2 phần tử)

- Khi n = 1: return(L) → **T(n) = C_1**

- Khi n > 1: gọi đệ quy **mergeSort** 2 lần cho 2 danh sách con với độ dài n/2 → T(n) = 2T(n/2)

- Sau khi có kết quả từ việc gọi đệ quy còn phải trộn hai danh sách kết quả (hàm **merge**): cần thời gian O(n) = nC_2.

- Vậy ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} C_1 & nếu\ n = 1 \\ 2T\left(\frac{n}{2}\right) + nC_2 & nếu\ n > 1 \end{cases}$$ O(nlogn)

VD3: Tháp Hà Nội, tính tổ hợp chập k của n

$$T(n) = \begin{cases} C_1 & nếu\ n = 1 \\ 2T(n-1) + C_2 & nếu\ n > 1 \end{cases}$$ $O(2^n)$

VD4: Tìm kiếm nhị phân dãy n có thứ tự

$$T(n) = \begin{cases} C_1 & nếu\ n = 1 \\ T\left(\frac{n}{2}\right) + C_2 & nếu\ n > 1 \end{cases}$$ O(logn)

(2) Giải phương trình đệ quy tìm nghiệm

→ Suy ra tỷ suất tăng f(n) hay O(f(n))

*** Có 3 phương pháp giải phương trình đệ quy:**

(1) Phương pháp truy hồi. (có thể sử dụng cho đệ quy lùi/ chia để trị)

- Triển khai T(n) theo T(n - 1), rồi T(n - 2), … cho đến T(1) hoặc T(0)

- Suy ra nghiệm

- Dùng đệ quy để thay thế T(m) với m < n (vào phía phải chương trình) cho đến khi tất cả T(m) với m > 1 được thay thế bởi biểu thức của T(1) hoặc T(0)

- Vì T(1) và T(0) là hằng số nên công thức T(n) chứa các số hạng chỉ

liên quan đến n và hằng số.

- Từ công thức đó suy ra nghiệm của phương trình.

VD1:

$$\boldsymbol{T(n) = \begin{cases} C_1 & \text{nếu } n = 0 \\ T(n-1) + C_2 & \text{nếu } n > 0 \end{cases}}$$

Ta có: T(n) = T(n-1) + C_2 (1)

= [T((n-1)-1) + C_2] + C_2 = T(n-2) + $2C_2$ (2)

= [T((n-2)-1) + C_2] + $2C_2$ = T(n-3) + $3C_2$ (3)

…

T(n) = T(n-i) + iC_2 (i)

Quá trình truy hồi kết thúc khi n-i = 0 hay i = n

$\Longrightarrow$ T(n) = T(0) + nC_2 = C_1 + nC_2 = O(**max**(1, n)) = **O(n)**

VD2:

$$\boldsymbol{T(n) = \begin{cases} C_1 & \text{nếu } n = 1 \\ 2T\left(\frac{n}{2}\right) + nC_2 & \text{nếu } n > 1 \end{cases}}$$

Ta có: T(n) = 2T(n/2) + nC_2 (1)

= 2[2T(n/4) + (n/2)C_2] + nC_2 =4T(n/4) + $2nC_2$ (2)

= 4[2T(n/8) + (n/4)C_2] + $2nC_2$ =8T(n/8) + $3nC_2$ (3)

= …

= 2^i T(n/2^i) + inC_2

Quá trình truy hồi kết thúc khi n/2^i = 1 hay 2^i = n → i = logn

$\Longrightarrow$ T(n) = nT(1) + logn.nC_2 = nC_1 + nlognC_2 =O(max(n, nlogn)) = **O(nlogn)**

*** Công thức cấp số nhân:** Nếu giá trị đầu là 1 và hệ số nhân là q thì tổng dãy i phần tử đầu là

$$\mathbf{S = \frac{q^i - 1}{q - 1}}$$

*** Công thức cấp số cộng:** Tính tổng n số hạng đầu

$$S = \frac{n(u_1 + u_n)}{2}$$

*** Một số công thức logarit:** $0 < N_1, N_2, N$ và $0 < a, b \neq 1$ ta có

- $log_a N = M \Leftrightarrow N = a^M$
- $log_a\left(\frac{N_1}{N_2}\right) = log_a N_1 - log_a N_2$
- $log_a a^M = M$
- $log_a N^\alpha = \alpha\ log_a N$
- $a^{log_a N} = N$
- $log_{a^\alpha} N = \frac{1}{\alpha}\ log_a N$
- $N_1^{log_a N_2} = N_2^{log_a N_1}$
- $log_a N = \frac{log_b N}{log_b a}$
- $log_a(N_1 N_2) = log_a(N_1) + log_a(N_2)$
- $log_a b = \frac{1}{log_b a}$

(2) Phương pháp đoán nghiệm (không học)

(3) Phương pháp lời giải tổng quát (có thể sử dụng cho chia để trị)

*** Giải thuật chia để trị:**

- Phân rã bài toán kích thước n thành a bài toán con có kích thước n/b.
- Giải các bài toán con và tổng hợp kết quả

* Với các bài toán con, tiếp tục *Chia để trị* đến khi các bài toán con kích thước 1 → *Thuật toán đệ quy.*

*** Giả thuyết:**

- Bài toán con kích thước 1 lấy **một** đơn vị thời gian
- Thời gian chia bài toán và tổng hợp kết quả để được lời giải của bài toán ban đầu là **d(n).**

*** Có thể áp dụng cho bài toán có dạng phương trình đệ quy tổng quát như sau:**

$$T(n) = \begin{cases} 1 & nếu\ n = 1 \\ aT\left(\frac{n}{b}\right) + d(n) & nếu\ n > 1 \end{cases}$$

Chú thích: a số lượng bài toán con, n/b kích thước bài toán con

- Với giả thuyết $n = b^k$, biến đổi công thức, ta được

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

+ Nghiệm thuần nhất: $a^k = n^{log_b a}$

+ Nghiệm riêng:

- **Nếu d(n) là hàm nhân (f(a.b) = f(a).f(b))**

Trường hợp 1: $\boldsymbol{a > d(b) \rightarrow T(n) = O(n^{log_b a})}$ **→ Cải tiến bằng cách giảm a**

Trường hợp 2: $\boldsymbol{a < d(b) \rightarrow T(n) = O\left(n^{log_b d(b)}\right)}$ **→ Cải tiến bằng cách giảm d(b)**

Trường hợp 3: $\boldsymbol{a = d(b) \rightarrow T(n) = O(n^{log_b a}\, log_b n)}$

- **Nếu d(n) không phải là hàm nhân:** tính thủ công

- Sau đó ta lấy MAX(NR, TNT)

VD1: GPT với T(1) = 1 và

1. $T(n) = 4T\left(\frac{n}{2}\right) + n$

- Phương trình đã cho có dạng phương trình đệ quy tổng quát

- d(n) = n là hàm nhân

- a = 4 và b = 2

- d(b) = d(2) = 2 < a = 4 (Trường hợp 1)

$\rightarrow \boldsymbol{T(n) = O(n^{log_b a}) = O(n^{log 4}) = O(n^2)}$

2. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

- Phương trình đã cho có dạng phương trình đệ quy tổng quát

- d(n) = n^2 là hàm nhân

- a = 4 và b = 2.

- d(b) = d(2) = $2^2 = 4$ = a (Trường hợp 3)

$\rightarrow \boldsymbol{T(n) = O(n^{log_b a}.log_b n) = O(n^{log 4} log n) = O(n^2 log n)}$

3. $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

Phương trình đã cho là chương trình đệ quy tổng quát

d(n) = n^3 là hàm nhân

a = 4, b = 2

d(b) = d(2) = $2^3 = 8$ > a = 4 (Trường hợp 2)

$\rightarrow \boldsymbol{T(n) = O\left(n^{log_b d(b)}\right) = O(n^{log 8}) = O(n^3)}$

4. $T(n) = 3T\left(\frac{n}{2}\right) + n$

- Phương trình đã cho có dạng phương trình tổng quát
- d(n) = n là hàm nhân
- a = 3 và b = 2
- d(b) = d(2) = 2 < a = 3 (Trường hợp 1)

→ $\boldsymbol{T(n) = O(n^{log_b a}) = O(n^{log 3})}$

5. $T(n) = 2T\left(\frac{n}{2}\right) + nlogn$

- Phương trình đã cho có dạng phương trình tổng quát
- d(n) = nlogn không phải là hàm nhân
- a = 2 và b = 2

* Nghiệm thuần nhất: $a^k = O(n^{log_b a}) = O(n^{log 2}) = O(n)$

* Nghiệm riêng = $\sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j \, 2^{k-j} \log 2^{k-j}$

$= \sum_{j=0}^{k-1} 2^j \, \frac{2^k}{2^j} \log 2^{k-j} = \sum_{j=0}^{k-1} 2^j \, \frac{2^k}{2^j} \, (k-j)$

$= 2^k \sum_{j=0}^{k-1} (k-j) = 2^k \frac{k(k+1)}{2} =$ O($2^k k^2$)

Do $n = b^k \rightarrow n = 2^k$ nên $k = logn$

→ NR = O($2^k k^2$) = O($2^{logn} (logn)^2$) = O(n$(logn)^2$) > O(n)

→ $\boldsymbol{T(n) = O(max(NR, NTN)) = O(n(logn)^2)}$

6. $T(n) = 2T\left(\frac{n}{2}\right) + logn$

- Phương trình có dạng đệ quy tổng quát

d(n) = logn không phải là hàm nhân

a = 2, b = 2

Nghiệm thuần nhất = $a^k = O(n^{log_b a}) = O(n^{log 2}) = O(n)$

Nghiệm riêng = $\sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j \log 2^{k-j}$

$= \sum_{j=0}^{k-1} 2^j \, (k-j) = \sum_{j=0}^{k-1} 2^j k - \sum_{j=0}^{k-1} 2^j \, j$

$=$ O($\sum_{j=0}^{k-1} 2^j \, k$) = O(k 2^k)

Do $n = b^k \rightarrow n = 2^k$ nên $k = logn$

→ NR = O($k\ 2^k$) = O($logn\ 2^{logn}$) = O(nlogn) > O(n)

→ $\boldsymbol{T(n) = O(max(NR, NTN)) = O(nlogn)}$

CHƯƠNG 2: SẮP XẾP

- Trình bày ý tưởng và minh họa các bước sắp xếp dãy số bằng phương pháp sắp xếp đơn giản: ***chọn, xen, nổi bọt.***

- Trình bày cách chọn phần tử khóa (pivot), cách phân hoạch và các bước sắp xếp dãy số bằng phương pháp ***sắp xếp nhanh*** (Quicksort).

- Trình bày quy tắc Pushdown và minh họa cách sắp xếp dãy số bằng phương pháp ***sắp xếp vun đống*** (Heapsort) dạng mảng

I. Các giải thuật sắp xếp cơ bản O(n^2)

- Sắp xếp chọn (Selection Sort)

- Sắp xếp xen (Insertion Sort)

- Sắp xếp nổi bọt (Bubble Sort)

1. Sắp xếp chọn **for _____ for**

(i: 0 → n-2) (j: i+1 → n-1)

* Lưu ý: Chiều hiển thị ở góc trên bên phải

* Nhận xét: - Độ phức tạp O(n^2)

- Không hề phụ thuộc vào tính chất dữ liệu nhập

- Thằng i = n-1 khỏi cần chọn

* Cách nhớ: lấy số nhỏ nhất trong dãy và hoán đổi với index i trong dãy, chỉ giữa 2 số với nhau, nếu có nhiều số cùng nhỏ nhất thì lấy số nhỏ nhất đầu tiên

```
void SelectionSort(int A[], int n){
    int i, j, lowindex, temp;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (A[j] < A[min])  min = j;
        }
        temp = A[i];  A[i] = A[min];   A[min] = temp;
    }
}
```

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	2	6	5	2	10	12	9	10	9	3
Bước 2		2	5	6	10	12	9	10	9	3
Bước 3			3	6	10	12	9	10	9	5
Bước 4				5	10	12	9	10	9	6
Bước 5					6	12	9	10	9	10
Bước 6						9	12	10	9	10
Bước 7							9	10	12	10
Bước 8								10	12	10
Bước 9									10	12
Kết quả	2	2	3	5	6	9	9	10	10	12

2. Sắp xếp xen for_____while

(i: 1 → n-1)

* Lưu ý: Chiều hiển thị ở góc dưới bên trái

* Nhận xét: - Độ phức tạp $O(n^2)$

- Phụ thuộc tính chất dữ liệu nhập

* Cách nhớ: lấy phần tử ở vị trí i đem nhét vào vị trí thích hợp của một danh sách chỉ đang gồm i + 1 phần tử, nếu ở gần đó có cùng số thì số mới được nhét ở cuối cùng

```
void InsertionSort(int A[], int n){
    int i, j;
    for(i = 1; i < n; i++){
        j = i;
        while(j > 0 && (A[j-1] > A[j])){
            Swap(A[j-1], A[j]);
            j--;
        }
    }
}
```

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	**3**
Bước 1	5	**6**								
Bước 2	**2**	5	6							
Bước 3	2	**2**	5	6						
Bước 4	2	2	5	6	**10**					
Bước 5	2	2	5	6	10	**12**				
Bước 6	2	2	5	6	**9**	10	12			
Bước 7	2	2	5	6	9	10	**10**	12		
Bước 8	2	2	5	6	9	**9**	10	10	12	
Bước 9	2	2	**3**	5	6	9	9	10	10	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

3. Sắp xếp nổi bọt for_____for

(i: 0 → n-2) (j: n-1 → i+1)

* Nhận xét: - Chiều hiển thị ở góc trên bên trái

- Không phụ thuộc tính chất dữ liệu nhập

- Các phần tử nhỏ được đưa về vị trí đúng rất chậm trong khi các phần tử lớn lại được đưa về vị trí đúng rất nhanh

```
void BubbleSort(int A[], int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = n - 1; j >= i + 1; j--){
            if(A[j-1] > A[j]){
                Swap(A[j-1], A[j]);
            }
        }
    }
}
```

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	5	6	2	3	10	12	9	10	9
Bước 2		**2**	5	6	3	9	10	12	9	10
Bước 3			**3**	5	6	9	9	10	12	10
Bước 4				**5**	6	9	9	10	10	12
Bước 5					**6**	9	9	10	10	12
Bước 6						**9**	9	10	10	12
Bước 7							**9**	10	10	12
Bước 8								**10**	10	**12**
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

II. Các thuật toán sắp xếp phức tạp

1. Sắp xếp nhanh

- Ý tưởng thuật toán: "Chia để trị"

* Tìm chốt: Chọn giá trị **khóa lớn nhất** trong 2 phần tử có khóa **khác nhau đầu tiên** kể từ trái qua.

$>$, 2, $\neq$ $\rightarrow$ Độ phức tạp O(n)

* Phân hạch: dùng 2 "con nháy" L và R, trong đó

- L đi từ bên **trái** và R đi từ bên **phải.**

• Cho L chạy **sang phải** tới khi gặp phần tử có **khóa ≥ chốt**

• Cho R chạy **sang trái** tới khi gặp phần tử có **khóa < chốt**

• Tại chỗ dừng của L và R: nếu L < R thì hoán vị a[L], a[R].

• Lặp lại quá trình dịch sang phải, sang trái của 2 "con nháy" L và R cho đến khi L > R.

• Khi đó L sẽ là điểm phân hoạch, cụ thể là a[L] là phần tử đầu tiên của mảng con "bên phải".

L	R	
< v	≥ v	Độ phức tạp O(n)

L < R: swap

L > R: Stop – Phân hạch tại L

* Trường hợp xấu nhất: Phân hạch lệnh: Xảy ra khi chốt là giá trị lớn nhất

$$\mathbf{T(n)} = \begin{cases} \mathbf{1} & \boldsymbol{n = 1} \\ \boldsymbol{T(n-1) + T(1) + n} & \boldsymbol{n > 1} \end{cases} \rightarrow O(n^2)$$

* Trường hợp trung bình: Phân hạch đều

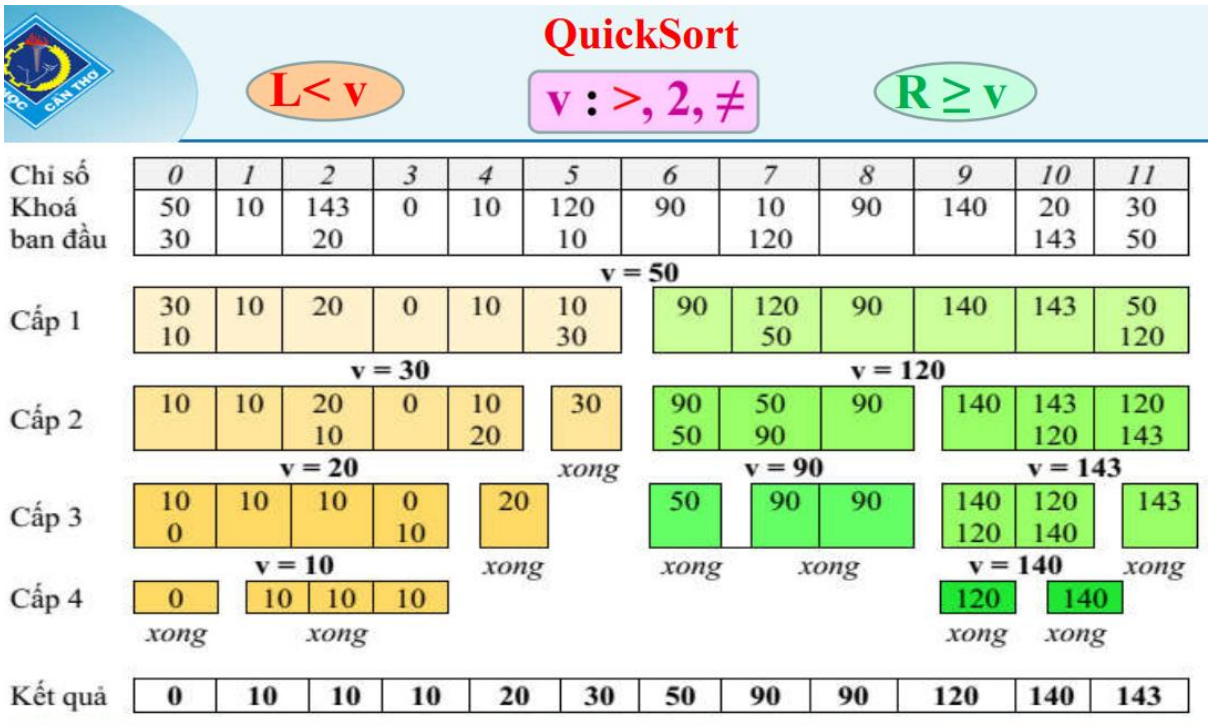
$$\mathbf{T(n)} = \begin{cases} \mathbf{1} & \boldsymbol{n = 1} \\ \boldsymbol{2T\left(\frac{n}{2}\right) + n} & \boldsymbol{n > 1} \end{cases} \rightarrow O(nlogn)$$

* QuickSort biến thể: Có vào trong thi

L < v: v: >, 2, ≠ R ≥ v

L ≤ v: v: < , 2, ≠ R > v

* Lưu ý: đệ quy sẽ dừng khi không tìm thấy chốt



* VD: QuickSort biến thể

Chỉ số	0	1	2	3	4	5	6	7	8	9	10	11
Ban đầu	50 10	10	143 10	0	10 143	120	90	10 50	90	140	20	30

v = 10

10 0	10	10	0 10	143 30	120	90	50	90	140 20	20 140	30 143

v = 0 v = 120

0	10	10	10	30	120 20	90	50	90	20 120	140	143		

Xong xong v = 30 v= 140

30 20	20 30	90	50	90	120	140	143				

v=20 xong xong

…

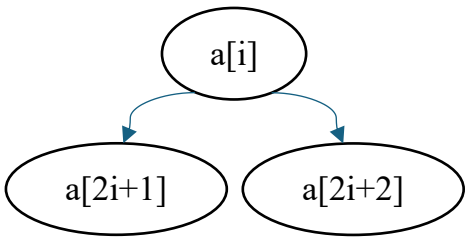
2. Sắp xếp vun đống

* Heap: Cây sắp thứ tự bộ phận $\begin{cases} \text{cây nhị phân} \\ \text{Giá trị nút (khác nút lá)không} \\ \text{lớn hơn giá trị các nút con} \end{cases}$

* HeapSort:

(1) Chuyển mảng thành cây nhị phân

- a[0] nút gốc



- Nút trong:
a[0] → a[(n-2)/2]

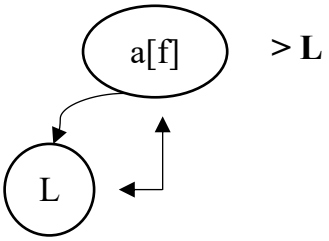
$\begin{cases} \text{n chẵn: a} \left[\frac{n-2}{2}\right] \text{ có L} \\ \text{n lẻ: a} \left[\frac{n-2}{2}\right] \text{ có L, R} \end{cases}$

(2) Chuyển cây nhị phân thành Heap

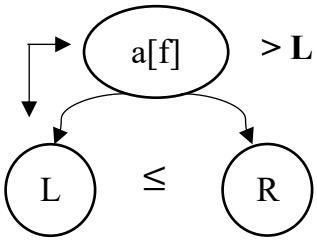
→ Thủ tục PushDown **O(logn)**

*** Có 3 trường hợp:**

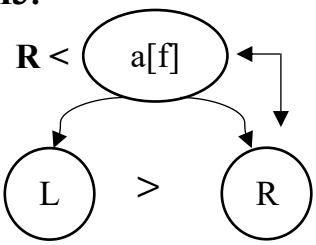
TH1:



TH2:



TH3:



*** Chú ý:** Khi gọi PushDown mà có sự hoán đổi vị trí giữa nút cha và nút nhánh thì tiếp tục lặp lại là xem xét đến nút con đã bị hoán đổi đó, cho đến khi tới lá.

→ Thuật toán HeapSort **O(nlogn)**

a) Tạo heap ban đầu

a[(n-2)/2] → a[0] gọi hàm PushDown

b) i: n-1 → 2

Swap(a[0],a[i])

Tạo heap con a[0] → a[i-1]: gọi PushDown

c) Swap(a[0], a[1])

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	**5** 2	**6** 2 5 3	**2**	**2** 6	**10** 3 5	**12**	**9**	**10**	**9**	**3** 10
Tạo Heap	**2** 10 2	**3**	**2** 10 9	**6**	**5**	**12**	**9** 10	**10**	**9**	**10** 2
i = 9	**2** 9 3	**3** 9 5	**9**	**6**	**5** 9	**12**	**10**	**10**	**9** 2	**2**
i = 8	**3** 10 5	**5** 10 6	**9**	**6** 10	**9**	**12**	**10**	**10** 3	**2**	
i = 7	**5** 10 6	**6** 10 9	**9**	**10**	**9** 10	**12**	**10** 5	3		
i = 6	**6** 12 9	**9** 12 10	**9**	**10** 12	**10**	**12** 6	**5**			
i = 5	**9** 10 9	**10**	**9** 10	**12**	**10** 9	**6**				
i = 4	**9** 12 10	**10** 12	**10**	**12** 9	**9**					
i = 3	**10** 10	**12**	**10** 10	**9**						
i = 2	**10** 12	**12** 10	**10**							
Kết quả	**12**	**10**	**10**	**9**	**9**	**6**	**5**	**3**	**2**	**2**

CHƯƠNG 3: KĨ THUẬT THIẾT KẾ THUẬT TOÁN

- Trình bày ***ý tưởng, nhận xét ưu khuyết điểm*** của các thuật toán: *vét cạn, chia để trị, tham ăn, nhánh cận, quy hoạch động, quay lui, cắt tỉa alpha-beta*. Nêu ý tưởng áp dụng vào một vấn đề cụ thể.

- Giải bài toán **cái ba lô** bằng các kĩ thuật: ***tham ăn, nhánh cận.***

- Trình bày các quy tắc ***cắt tỉa alpha-beta*** và áp dụng định trị nút gốc trên **cây trò chơi**

I. Kĩ thuật chia để trị

*** Ý tưởng:** Chia bài toán lớn thành các bài toán con nhỏ hơn, giải quyết từng bài toán con rồi kết hợp kết quả.

*** Ưu điểm:**

- Hiệu quả với các bài toán có cấu trúc chia nhỏ được (ví dụ: sắp xếp, tìm kiếm).

- Dễ dàng song song hóa.

*** Nhược điểm:**

- Không phù hợp với các bài toán không thể chia nhỏ.

(1) Quick Sort/ Merge Sort O(nlogn)

(2) Bài toán nhân 2 số nguyên lớn

- Nhân thông thường $O(n^2)$
- Chia để trị $O(n^2)$
- Chia để trị cải tiến $O(n^{log3} \approx n^{1.58})$

II. Kĩ thuật tham ăn (Greedy)

*** Ý tưởng:** Tại mỗi bước, chọn giải pháp tối ưu cục bộ với hy vọng đạt được giải pháp tối ưu toàn cục.

*** Ưu điểm:**

- Hiệu quả về thời gian và bộ nhớ.

- Dễ cài đặt.

- Thường dùng giải các bài toán tối ưu tổ hợp

*** Nhược điểm:**

- Không phải lúc nào cũng đảm bảo tìm được giải pháp tối ưu toàn cục.

$X = (x_1, x_2, \ldots, x_n), \forall X \in D \rightarrow f(X^*) \begin{cases} \text{Max} \\ \text{Min} \end{cases}$

→ Nếu dùng phương pháp "vét cạn" để tìm phương án tối ưu: cần một thời gian mũ

* Các bài toán:

(1) Bài toán trả tiền máy ATM f(X) → min

(2) Bài toán đường đi của người giao hàng f(X) → min

- Vét cạn:

→ Số chu trình: $\frac{(n-1)!}{2}$ (Kiểm tra trắc nghiệm) **O(n!)**

- Tham ăn:

+ Sắp xếp các cạnh theo thứ tự tăng của độ dài

+ Xét các cạnh có độ dài từ nhỏ đến lớn để đưa vào chu trình

+ Một cạnh sẽ được đưa vào chu trình nếu:

- Không tạo thành một chu trình thiếu
- Không tạo thành một đỉnh có cấp ≥ 3

+ Lặp lại bước (3) cho đến khi xây dựng được một chu trình.

→ Số phép chọn: $\frac{n(n-1)}{2}$ **O(n^2)**

(3) Bài toán cái ba lô

TL: W Max

ĐV: n → i $\begin{cases} \text{TL: } g_i \\ \text{GT: } v_i \\ \text{GT: } -- \end{cases}$ (CBL 1) ↓ Đơn giá = GT/TL

Lưu ý: Xuất phương án phải đúng với trình tự ban đầu

*** Biến thể của bài toán balô:**

ĐV: n → i $\begin{cases} \text{TL: } g_i \\ \text{GT: } v_i \\ \text{GT: } s_i \end{cases}$ (CBL 2)

ĐV: n → i $\begin{cases} \text{TL: } g_i \\ \text{GT: } v_i \\ \text{GT: } 1 \end{cases}$ (CBL 3)

Ví dụ: Cho bài toán cái ba lô với trọng lượng của ba lô **W = 30** và **5 loại đồ vật** được cho trong bảng bên dưới. Tất cả các đồ vật đều ***chỉ có 1 cái***. Giải bài toán bằng **kỹ thuật tham ăn** (Greedy)?

Loại đồ vật	**Trọng lượng**	**Giá trị**
A	**15**	**30**
B	**10**	**25**
C	**2**	**2**
D	**4**	**6**
E	**8**	**24**

W = 30 **CBL3**

ĐV	**ĐG**
E	3
B	2.5
A	2
D	1.5
C	1

- Phương án X = (x_A, x_B, x_C, x_D, x_E)

- $x_E = 30/8 = 3 > 1 \rightarrow x_E = 1$

W = 30 – 8*1 = 22

- $x_B = 22/10 = 2 > 1 \rightarrow x_B = 1$

W = 22 – 10*1 = 12

- $x_A = 12/15 = 0 < 1 \rightarrow x_A = 0$

W = 12

- $x_D = 12/4 = 3 > 1 \rightarrow x_D = 1$

W = 12 – 4*1 = 8

- $x_C = 8/2 = 4 > 1 \rightarrow x_C = 1$

W = 8 – 2*1 = 6

→ Phương án là X = (0, 1, 1, 1, 1)

* Tổng TL: 8 + 10 + 4 + 2 = 24

* Tổng GT: 25 + 2 + 6 + 24 = 57

Ví dụ: Bài toán mua hàng với số tiền **M = 40** và **5 loại hàng** hóa được cho trong bảng bên dưới. Giải bài toán bằng **kỹ thuật tham ăn** (Greedy)?

Loại hàng	Giá tiền	Giá trị dinh dưỡng	Số lượng
A	10	25	2
B	15	30	1
C	4	6	1
D	2	2	2
E	8	24	1

M = 40 **CBL2**

Loại hàng	ĐG
E	3
A	2.5
B	2
C	1.5
D	1

- Phương án X = (x_A, x_B, x_C, x_D, x_E)

- $x_E = 40/8 = 5 > 1 \rightarrow x_E = 1$

W = 40 – 8*1 = 32

- $x_A = 32/10 = 3 > 2 \rightarrow x_A = 2$

W = 32 – 10*2 = 12

- $x_B = 12/15 = 0 < 1 \rightarrow x_B = 0$

W = 12

- $x_C = 12/4 = 3 > 1 \rightarrow x_C = 1$

W = 12 – 4*1 = 8

- $x_D = 8/2 = 4 > 2 \rightarrow x_D = 2$

W = 8 – 2*2 = 4

→ Phương án là X = (2, 0, 1, 2, 1)

* Tổng TL: 2*10 + 4 + 2*2 + 8= 36

* Tổng GT: 25*2 + 6 + 2*2 + 24 = 84

III. Kĩ thuật nhánh cận

*** Ý tưởng:** Duyệt có hệ thống các giải pháp tiềm năng, loại bỏ các nhánh không có khả năng dẫn đến giải pháp tối ưu.

*** Ưu điểm:**

- Giảm không gian tìm kiếm so với vét cạn.

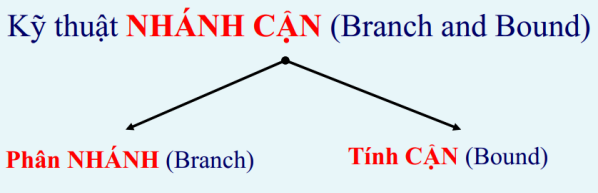
- Hiệu quả với các bài toán tối ưu hóa tổ hợp.

- Cho phép tìm được **phương án tối ưu.**

*** Nhược điểm:**

- Cần phải có cách *ước lượng giá trị cận tốt* mới có thể cắt được nhiều nhánh → hạn chế không gian tìm kiếm nhằm nhanh chóng xác định phương án tối ưu.

- Chi phí tính toán vẫn có thể lớn nếu không có cận tốt.



- *Nút gốc* biểu diễn cho tập tất cả các phương án có thể có

- *Nút lá* biểu diễn cho một phương án nào đó.

- *Nút trong* n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n.

→ Kỹ thuật này gọi là **phân nhánh**

* Với mỗi nút trên cây: tính một ***giá trị cận*** (giá trị gần với giá của các phương án).

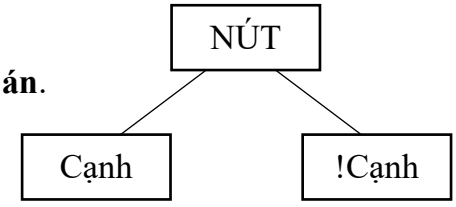
- **Bài toán tìm MIN**: xác định **cận dưới** (≤ giá phương án)

- **Bài toán tìm MAX**: xác định **cận trên** (≥ giá phương án)

→ Kì vọng tìm được một phương án tốt hơn

* Các bài toán:

(1) Bài toán TSP: f(x) → Min



- *Nút gốc*: biểu diễn cấu hình gồm **tất cả các phương án**.

- *Nút con **trái***: biểu diễn cấu hình gồm tất cả các phương án chứa một cạnh nào đó,

- *Nút con **phải***: biểu diễn cấu hình gồm tất cả các phương án **không** chứa cạnh đó (các cạnh được xét theo thứ tự, chẳng hạn thứ tự từ điển)

- *Mỗi nút* trong kế thừa các thuộc tính của tổ tiên nó và có thêm một thuộc tính mới

- *Nút lá:* biểu diễn cho cấu hình chỉ bao gồm một phương án

* Quy tắc để có thể phân nhánh tới nút lá:

- Mọi đỉnh trong chu trình đều có cấp 2

- Không tạo ra một chu trình thiếu.

* Ví dụ: Xét bài toán TSP có 5 đỉnh với độ dài các cạnh được cho như hình.

Bài toán tìm MIN → Tính cận dưới (CD)

a) Xây dựng nút gốc, tính cận dưới cho nút gốc.

b) Sau khi **phân nhánh** cho mỗi nút, **tính cận dưới** cho cả hai con.

Trong quá trình xây dựng cây có thể phát sinh một số nút lá (biểu diễn phương án). Giá nhỏ nhất trong số các phương án = ***giá nhỏ nhất tạm thời (GNNTT)***

c) Nếu **cận dưới của một nút con ≥** ***giá nhỏ nhất tạm thời*** : không cần xây dựng các cây con cho nút này nữa (Ta gọi là cắt tỉa các cây con của nút đó).

d) Nếu cả hai nút con đều có cận dưới < ***giá nhỏ nhất tạm thời*** : nút con nào có **cận dưới nhỏ hơn sẽ được ưu tiên phân nhánh trước** (vì **hy vọng** ở hướng đó sẽ có phương án tối ưu)

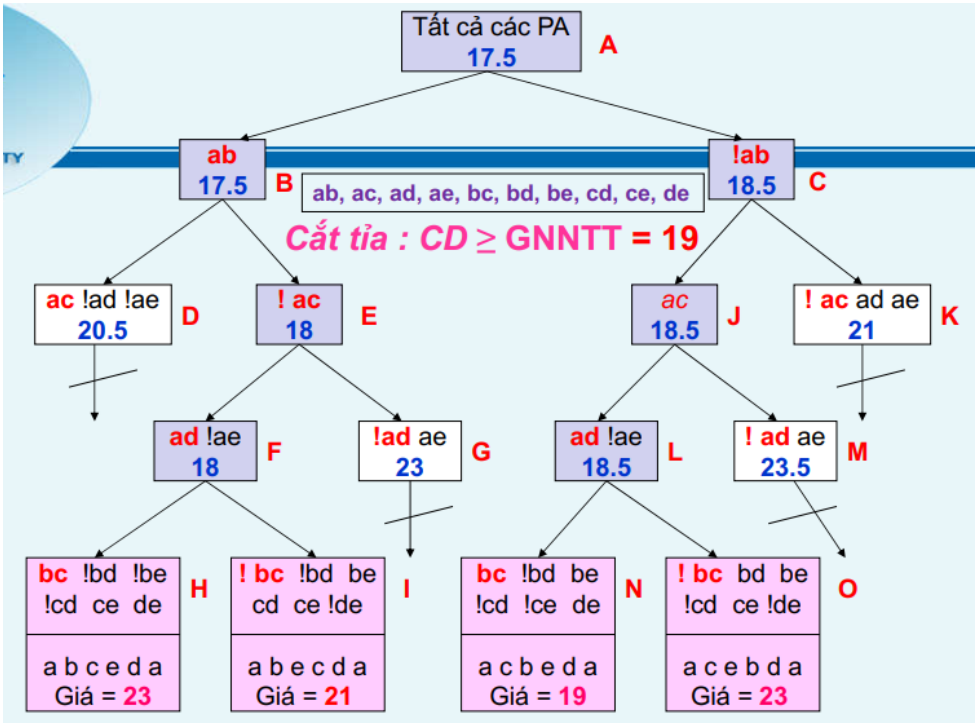
e) Mỗi lần **quay lui để xét nút con chưa được xét**: phải xem lại nút con đó để cắt tỉa các cây con của nó vì có thể phát sinh một ***giá nhỏ nhất tạm thời*** mới

Khi tất cả các con đã *được phân nhánh* hoặc *bị cắt tỉa* thì phương án có giá nhỏ nhất trong các phương án tìm được là phương án tối ưu cần tìm.

* Cách tìm cận dưới:

- Từ mỗi đỉnh lấy 2 cạnh có trọng số nhỏ nhất, tuy nhiên vẫn thỏa mãn các điều kiện được ghi trên nút đó.

* Cách cạnh theo thứ tự từ điển để xét là: **ab**, **ac**, ad, ae, bc, bd, be, cd, ce và de.



(2) Bài toán cái balo

Danh sách đồ vật: sắp xếp theo thứ tự giảm của đơn giá

a) Nút gốc biểu diễn trạng thái ban đầu của ba lô (chưa chọn một vật nào)

– Tổng giá trị được chọn: TGT = 0.

– Cận trên của nút gốc: CT = W * Đơn giá lớn nhất.

b) Nút gốc có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con, tính:

– TGT = TGT (nút cha) + số đồ vật được chọn * giá trị vật.

– W = W (nút cha) - số đồ vật được chọn * trọng lượng vật.

– CT = TGT + W * Đơn giá vật xét kế tiếp.

c) Trong các nút con, ưu tiên phân nhánh cho nút con có *cận trên lớn hơn* trước (Bài toán MAX). Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút, xác định lại các thông số TGT, W, CT theo công thức bước (2)

d) Lặp lại bước 3 với chú ý: đối với những nút có ***cận trên ≤ giá lớn nhất tạm thời (GLNTT)*** của phương án đã được tìm thấy thì không cần phân nhánh cho nút đó (cắt tỉa)

e) Nếu tất cả các nút đều đã được phân *nhánh* hoặc *bị cắt tỉa* thì phương án có giá lớn nhất là phương án cần tìm.

Ví dụ: Có một ba lô có trọng lượng là W= 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng bên dưới:

ĐV	**TL**	**GT**
A	15	30
B	10	25
C	2	2
D	4	6

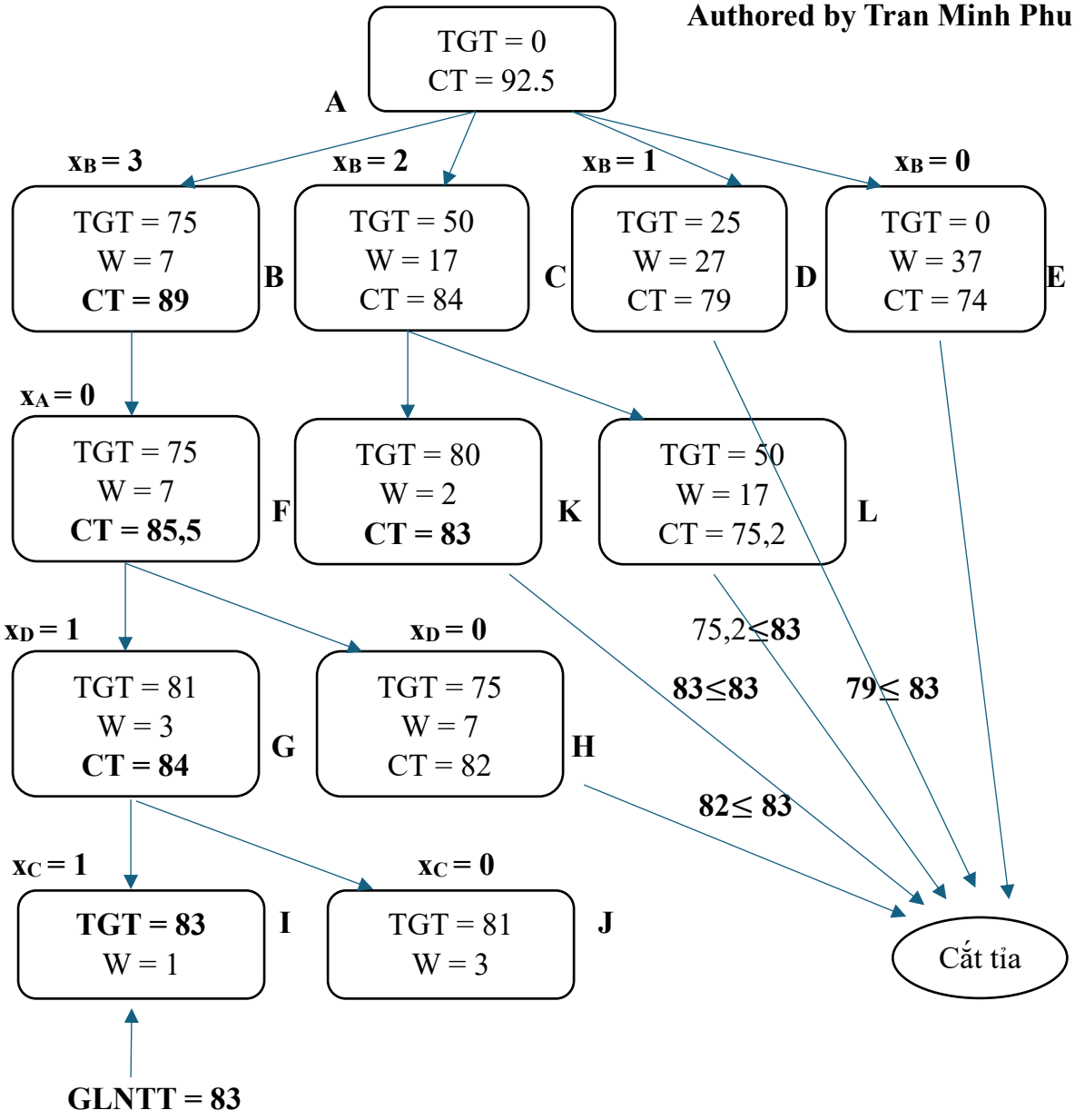
ĐV	**TL**	**GT**	**ĐG**
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

CBL1 W = 37, n = 4

Phương án $X = (x_A, x_B, x_C, x_D)$

CT_A = W * DG(Vật max)

= 37 * 2.5 = 92.5



Bài toán MIN: Xác định cận dưới Cắt khi <u>CD ≥ GNNTT</u>

Bài toán MAX: Xác định cận trên Cắt khi <u>CT ≤ GLNTT</u>

IV. Kĩ thuật quy hoạch động

*** Ý tưởng:** Trong thuật toán đệ quy, một số bài toán con phải giải nhiều lần

→ **Giải pháp:** Tạo bảng lưu trữ kết quả các bài toán con để khi cần sẽ sử dụng mà không cần phải giải lại

*** Ưu điểm:**

– Chương trình **thực hiện nhanh.**

– Kỹ thuật quy hoạch động có thể vận dụng để giải các bài toán *tối ưu*, **các bài toán có *công thức truy hồi***

*** Nhược điểm:** Quy hoạch động không hiệu quả khi:

– Không tìm được công thức truy hồi.

– Số lượng bài toán con cần giải quyết và lưu giữ kết quả là rất lớn.

– Sự kết hợp lời giải của các bài toán con chưa chắc cho lời giải của bài toán ban đầu

Thuật toán Quy hoạch động : 2 bước

***a)* Tạo bảng** bằng cách:

– Gán giá trị cho một số ô nào đó.

– Gán trị cho các ô khác nhờ vào giá trị của các ô trước đó.

***b)* Tra bảng** và xác định kết quả bài toán ban đầu.

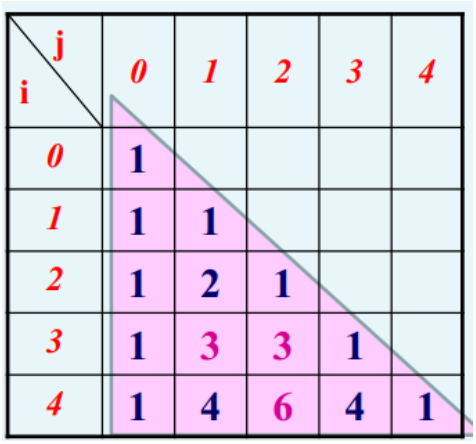
*** Các bài toán:**

(1) Bài toán tổ hợp chập k của n

- Đệ quy: $O(2^n)$

- Quy hoạch động

a) Tạo bảng: Xây dựng một bảng C gồm n+1 dòng (từ 0 đến n) và n+1 cột (từ 0 đến n).



• C[i,j] lưu trữ giá trị của Comb(i,j) theo quy tắc truy hồi sau:

(Quy tắc tam giác Pascal):

– C[0,0] = 1; – C[i,0] =1;

– C[i,i] = 1 với $0 < i \le n$;

– C[i,j] = C[i-1,j-1] + C[i-1,j] với $0 < j < i \le n$.

b) Tra bảng: C[n,k] chính là Comb(n,k)

- Thuật toán quy hoạch động gán trị cho ½ mảng hai chiều (n+1) dòng, (n+1) cột với số phần tử là $(n-1)^2/2$ ô của bảng

- $T(n) = (n-1)^2 C/2 = O(n^2)$ (hiệu quả hơn nhiều so với đệ quy)

(2) Bài toán cái ba lô

- Chỉ sử dụng được khi TL balo và TL đồ vật là số nguyên

V. Kĩ thuật cắt tỉa alpha-beta trên cây trò chơi

1. Kĩ thuật quay lui

* Là quá trình phân tích **đi xuống** và **quay lui lại** theo đường đã đi qua.

- Tại mỗi bước: vấn đề chưa được giải quyết (do còn thiếu cứ liệu) nên phải phân tích tới các **điểm dừng** - xác định được *lời giải/* xác định được *không thể (/không nên) tiếp tục đi*.

- Từ các **điểm dừng**: quay lui theo đường đã qua để giải quyết các vấn đề tồn đọng ⇒ giải quyết vấn đề ban đầu

*** 3 kĩ thuật quay lui:**

- "Vét cạn": đi tới tất cả các điểm dừng rồi mới quay lui

- "**Cắt tỉa Alpha-Beta"** và **"nhánh cận":** không cần thiết phải đi tới tất cả các điểm dừng, chỉ đến một số điểm và suy luận để quay lui sớm

*** Bài toán cây biểu thức số học: Mô tả**

Cây biểu thức số học là cây nhị phân:

- *Nút lá* biểu diễn toán hạng

- *Nút trong* biểu diễn toán tử

```
float Eval (node n) {
    if (n là nút lá)
        return (trị của toán hạng trong n);
    return (Toán tử trong n (Eval (Con trái của n), Eval (Con phải của
n)));
}
```

2. Cây trò chơi

*** Ý tưởng:** Cải tiến thuật toán Minimax bằng cách loại bỏ các nhánh không cần thiết trong cây tìm kiếm.

*** Ưu điểm:**

- Giảm đáng kể số lượng nút cần duyệt.

- Hiệu quả với các trò chơi có luật chặt chẽ (ví dụ: cờ vua, cờ tướng).

*** Nhược điểm:**

- Phụ thuộc vào thứ tự duyệt các nút.

- Xét trò chơi có 2 người luân phiên đi nước cờ của mình.

- Trò chơi có một *trạng thái ban đầu* và mỗi nước đi sẽ biến đổi *trạng thái hiện hành* thành *trạng thái mới*

- Trò chơi kết thúc khi : có một người **thắng cuộc** hoặc cả hai đấu thủ không thể phát triển được nước đi của mình (**hòa cờ**).

- Vấn đề: Tìm cách phân tích xem từ một trạng thái nào đó sẽ dẫn đến đấu thủ nào thắng với điều kiện cả hai đấu thủ đều có trình độ ngang nhau → Cây trò chơi

* Cách biểu diễn trò chơi bằng cây trò chơi:

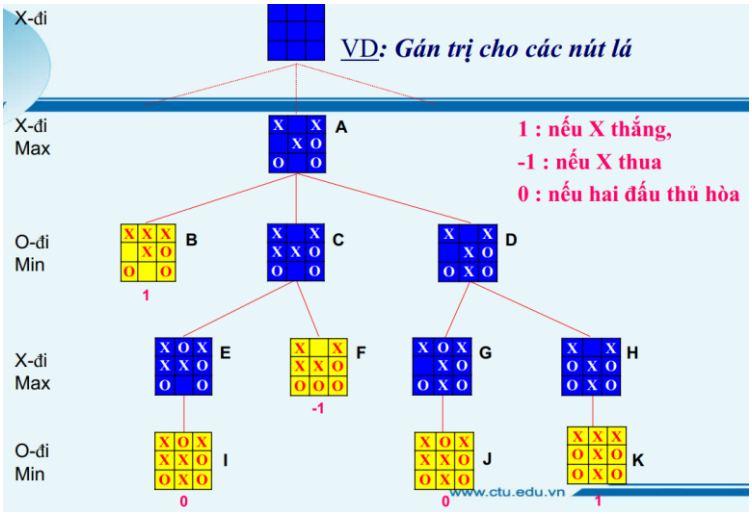
- Trò chơi có thể được biểu diễn bởi cây trò chơi.

- Mỗi nút của cây biểu diễn cho một trạng thái

+ **Nút gốc:** biểu diễn cho trạng thái bắt đầu cuộc chơi

+ **Mỗi nút lá:** biểu diễn cho một trạng thái kết thúc của trò chơi (trạng thái thắng, thua, hoặc hòa)

- Nếu trạng thái x được biểu diễn bởi nút n thì các con của n biểu diễn cho tất cả các trạng thái kết quả của các nước đi có thể xuất phát từ trạng thái



- Lượt đi của X: X chọn nước đi dẫn đến trạng thái có giá trị **lớn nhất (1)** hay X chọn nước đi MAX → nút **MAX**.

- Lượt đi của O: O chọn nước đi dẫn đến trạng thái có giá trị **nhỏ nhất (-1)** (X thua, O thắng) hay O chọn nước đi MIN → nút **MIN**.

Do 2 đấu thủ đi luân phiên nhau nên các mức trên cây cũng luân phiên là MAX và MIN → *Cây trò chơi* còn gọi là ***cây MIN-MAX***

*** Quy tắc định trị:**

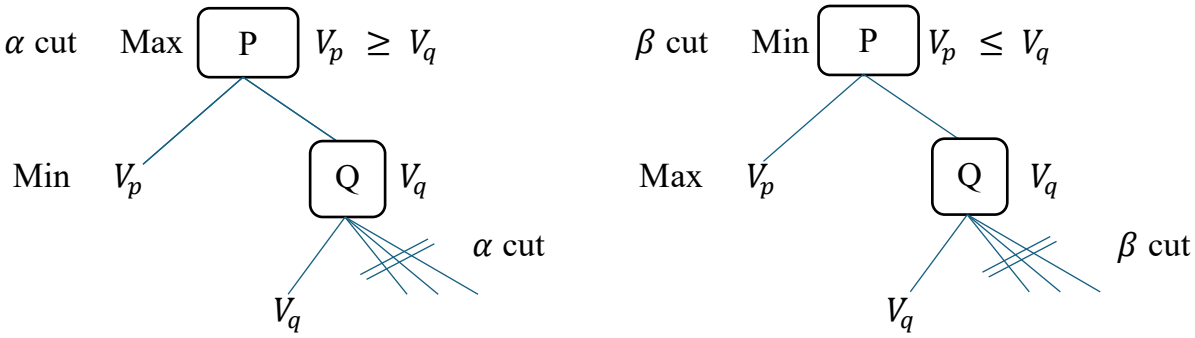
- **Nút lá** thì *trị của nó = giá trị được gán cho **nút đó***.

Ngược lại:

- Nếu nút là **nút MAX** thì *trị của nó = giá trị **lớn nhất** (Max)* của tất cả các trị của các con của nó.

- Nếu nút là **nút MIN** thì *trị của nó = giá trị **nhỏ nhất** (Min)* của tất cả các trị của các con của nó.

*** Kĩ thuật vét cạn:** Tìm cách sao cho khi định trị một nút thì ***không nhất thiết*** phải định trị cho tất cả các nút con cháu của nó



*** Quy tắc định trị: cho một nút *không phải là nút lá* :**

– Khởi đầu nút MAX có giá trị tạm -∞; nút MIN là ∞.

– Nếu tất cả các nút con của một nút ***đã được xét*** hoặc ***bị cắt tỉa*** thì giá trị tạm của nút đó trở thành giá trị của nó.

– Nếu một nút MAX n có giá trị tạm là V1 và một nút con của nó có giá trị là V2 thì đặt giá trị tạm mới của n là **max(V1,V2)**. Nếu n là nút MIN thì đặt giá trị tạm mới của n là **min(V1,V2)**.

⇒ Vận dụng **quy tắc cắt tỉa Alpha-Beta** để hạn chế số lượng nút phải xét.

CHƯƠNG 4: LƯU TRỮ NGOÀI

- Định nghĩa và các tính chất của B-cây.
- Thực hiện xen, xóa mẩu tin trong tập tin dạng B-cây

I. Mô hình xử lý ngoài

II. Cách tổ chức tập tin ngoài

1. Định nghĩa

- B cây bậc m là cây tìm kiếm m phân cân bằng

2. Tính chất

(1) Nút gốc hoặc là lá hoặc có 2 nút con

(2) Mỗi nút, trừ nút gốc và nút lá, có từ [m/2] → m nút con

(3) Các đường đi từ gốc đến lá có cùng độ dài

(4) Các khóa và cây con được sắp xếp theo cây tìm kiếm

3. Các thao tác

a) Xen

VD: Xen 37

- Quá trình tìm kiếm bắt đầu từ gốc → P_2 → L_7

- Do L_7 hết chỗ nên cấp lá mới L_7' dời [b/2] = 2 mẫu tin cuối (36, 38) → L_7', xen 37 → L_7', xen L_7' → P_2

- Do P_2 hết chỗ nên cấp nút mới P_2' dời [m/2] = 3 lá cuối (L_7, L_7', L_8) → P_2'

- Gốc còn chỗ: xen P_2' vào

- Cập nhật lại khóa → B-cây sau

b) Xóa

VD: Xóa 12

- Quá trình tìm kiếm bắt đầu từ gốc → P_1 → L_3

- Xóa 12 khỏi L_3 → L_3 rỗng → giải phóng L_3

- Xóa cặp khóa con trỏ của L_3 trong P_1, P_1 chỉ còn 2 con (< [m/2])

- Xét P_2 bên phải cùng mức với P_1, P_2 có 5 con (> [m/2]), nên chuyển con trái nhất L_4 → P_1

(- Hoặc xét P_2 bên phải cùng mức với P_1, P_2 có 3 con (= [m/2]), nên nối P_2 vào P_1, giải phóng P_2)

- Cập nhật lại khóa → B-cây sau