Giáo trình Lý thuyết thuật toán-Bộ môn Khoa học máy tính-2010

……….., tháng … năm …….

MỤC LỤC

Nội dung	Trang
Chương 1: Kỹ thuật phân tích đánh giá thuật toán	4
1.1. Khái niệm bài toán và độ phức tạp dữ liệu vào	4
1.1.1. Khái niệm bài toán	4
1.1.2. Độ phức tạp dữ liệu vào của bài toán	4
1.2. Các mô hình tính toán	4
1.2.1. Máy Turing	5
1.2.2. Máy xử lý thuật toán viết bằng ngôn ngữ tựa ALGOL	7
1.3. Khái niệm thuật toán và độ phức tạp của thuật toán	7
1.3.1. Thuật toán(*Algorithm*)	7
1.3.2 Chi phí phải trả cho một quá trình tính toán và các khái niệm về độ phức tạp thuật toán	7
1.4. Cách tính độ phức tạp	10
1.4.1. Các quy tắc cơ bản	10
1.4.2. Độ phức tạp của các chương trình đệ quy	11
1.5. Thuật toán không đơn định đa thức(Nodeterministic Polynomial NP)	14
1.5.1. Sự phân lớp các bài toán.	16
1.5.2. Khái niệm "dẫn về được" (*Phép quy dẫn*)	17
1.5.3 Lớp bài toán NP - khó (*NP - hard*) và NP - đầy đủ (*NP – Complate*)	18
1.6. Thuật toán xấp xỉ (*Heuristic*)	22
1.6.1. Các khái niệm	22
1.6.2. Thuật toán ε - xấp xỉ tuyệt đối	24
1.6.3.Thuật toán ε - xấp xỉ	26
1.7. Chứng minh tính đúng đắn của thuật toán	28
1.7.1. Ví dụ:	28
1.7.2. Phương pháp thử	28
1.7.3. Kiểm chứng tính đúng đắn	29
Chương 2: Các thuật toán Sắp xếp	31
2.1. Bài toán sắp xếp	31
2.1.1. Tầm quan trọng của bài toán sắp xếp	31
2.1.2. Sắp xếp trong và sắp xếp ngoài	31

2.1.3. Tổ chức dữ liệu và ngôn ngữ cài đặt 31

2.1.4. Thuật toán sắp xếp 32

2.2. Các phương pháp sắp xếp đơn giản 32

2.2.1. Sắp xếp chọn (Selection Sort) 32

2.2.2. Sắp xếp chèn (InsertionSort) 33

2.2.3. Sắp xếp nổi bọt(Bubble Sort) 35

2.3. Sắp xếp nhanh QUICK SORT 36

2.3.1. Ý tưởng 36

2.3.2. Thiết kế giải thuật 36

2.3.3. Đánh giá độ phức tạp 38

2.4. Sắp xếp kiểu vun đống (Heapsort) 39

2.4.1. Định nghĩa HEAP 39

2.4.2. Sắp xếp kiểu vun đống 40

2.4.3. Độ phức tạp tính toán 40

2.5. Sắp xếp hòa nhập (*Merge Sort*) 43

2.5.1. Ý tưởng 43

2.5.2. Thiết kế giải thuật 44

2.5.3. Đánh giá độ phức tạp 46

2.6. Cấu trúc dữ liệu và giải thuật xử lý ngoài 46

2.6.1. Mô hình xử lý ngoài 46

2.6.2. Đánh giá các giải thuật xử lý ngoài 47

2.6.3. Sắp xếp ngoài - MergeSorting 47

2.6.4. Cải tiến sắp xếp trộn 51

2.6.5. Trộn nhiều đường 52

Chương 3: Kỹ thuật thiết kế thuật toán 54

3.1. Chia để trị 54

3.1.1. Nội dung 54

3.1.2. Một số bài toán áp dụng 55

3.2. Quy hoạch động *(Dynamic)* 58

3.2.1. Nội dung 58

3.2.2. Ví dụ áp dụng 59

3.3. Phương pháp tham lam (*Greedy Method*) 63

3.3.1. Bài toán tối ưu tổ hợp 63

3.3.2. Nội dung 64

3.4. Phương pháp nhánh cận *(Branch- and- Bound*) 68

3.4.1. Nội dung 68

3.4.2. Các bài toán áp dụng 69

Chương 4: Lý thuyết lập lịch 75

4.1. Vấn đề lập lịch tối ưu 75

4.1.1. Bài toán 75

4.1.2. Nhận xét 76

4.1.3. Tình hình giải bài toán lập lịch hiện nay 77

4.2. Phân lớp bài toán lập lịch dạng tĩnh 78

4.2.1. Thông tin về công việc 78

4.2.2. Quan hệ giữa các máy 78

4.2.3. Quan hệ giữa các công việc 79

4.2.4. Một số tiêu chuẩn tối ưu 80

4.2.5. Một số ví dụ 80

4.2.6. Một số thuật toán lập lịch 81

4.3. Một số bài toán lập lịch giải bằng thuật toán lập lịch tối ưu nhanh 81

4.3.1. Hệ $1,n||C_{max}$ 81

4.3.2. Nhóm hệ $1, n || L_{max}$ và $1, n || T_{max}$ 83

4.3.3. Hệ $1,n| r_i \geq 0 |C_{max}$ 85

4.4. Bài toán lập lịch gia công trên 2 máy, thuật toán Johnson 88

4.4.1. Bài toán $2; F|r_i \geq 0| C_{max}$ 88

4.4.2. Thiết kế thuật toán 88

4.4.3. Một số trường hợp riêng có thể dẫn về bài toán 2 máy 91

Chương 1

KỸ THUẬT PHÂN TÍCH, ĐÁNH GIÁ THUẬT TOÁN

1.1. Khái niệm bài toán và độ phức tạp dữ liệu vào

1.1.1. Khái niệm bài toán

- Thông thường một bài toán được cho dưới dạng sau:

 + Input: Các dữ liệu vào của bài toán.

 + Output: Các dữ liệu ra thoả mãn yêu cầu của bài toán.

- Giải bài toán có nghĩa là xuất phát từ dữ liệu vào, thực hiện một dãy hữu hạn những thao tác có cơ sở khoa học thích hợp để tìm được dữ liệu ra (kết quả) theo yêu cầu của bài toán.

1.1.2. Độ phức tạp dữ liệu vào của bài toán

Có hai quan niệm chủ yếu:

Quan niệm 1(quan niệm đơn giản): Độ phức tạp dữ liệu vào của bài toán đựoc hiểu là số lượng dữ liệu vào của bài toán (kích thước của bài toán

Quan niệm 2: Là tổng độ dài của mọi dữ liệu vào đã được mã hóa theo một cách nào đó.

Ví dụ: Cho dãy số nguyên X={x_1,x_2,…,x_n}. Tìm giá trị lớn nhất trong dãy?

Bài toán được biểu diễn như sau:

Input : Cho dãy số nguyên X= {x_1,x_2,…,x_n}, số lượng n.

Output: Tìm số lớn nhất Max của dãy X.

- Theo quan niệm 1 : Kích thước của bài toán là (n+1)
- Theo quan niệm 2 : Kích thước của bài toán là

 + Số tự nhiên x_i theo mã nhị phân có độ dài là $[\log_2 x_i]+1$

VD: x_i	mã	độ dài
3	11	$[\log_2 3]+1=2$
5	101	$[\log_2 5]+1=3$

 +Độ dài dữ liệu của bài toán trên là: $\sum_{i=1}^{n} [\log_2 x_i] + \log_2 n + n + 1$

1.2. Các mô hình tính toán

Thông tường người ta xét đến 2 mô hình tính toán thông dụng:

- Mô hình lí thuyết: Máy Turing.

- Mô hình ứng dụng: Máy xử lý thuật toán viết bằng ngôn ngữ tựa Algol (các ngôn ngữ lập trình bậc cao).

1.2.1. Máy Turing

a) Câú tạo: + Bộ nhớ: Gồm một băng tuyến tính vô hạn ở đầu phải, chia thành các ô nhớ, mỗi ô chứa được một kí hiệu nguyên tố. n ô trái ($n \geq 0$) được ghi các kí hiệu của xâu vào, phần còn lại ở bên phải được lấp đầy bởi một kí hiệu đặc biệt gọi là kí hiệu trắng B.

+ Bộ điều khiển: Có hữu hạn trạng thái, tại mỗi thời điểm có một trạng thái xác định.

+ Một đầu đọc/ viết, nó cho phép tại một thời điểm có thể đọc hay viết ở một ô trên băng.

b) Hoạt động: Theo thời gian "rời rạc", được điều khiển bởi bộ điều khiển.

Tùy thuộc vào trạng thái hiện tại và kí hiệu đọc được trên băng mà nó tiến hành một bước chuyển gồm đồng thời 3 động tác sau:

1. Đổi trạng thái trên bộ điều khiển
2. Viết một kí hiệu lên ô đang đọc
3. Chuyển đầu đọc viết sang phải hay trái một ô theo quy định của hàm chuyển.

Một cách hình thức, xem máy Turing là một bộ $T = (\Sigma, Q, \Gamma, \delta, q_0, B, F)$

Trong đó :

Q: Tập hữu hạn các trạng thái.

Γ : Tập hữu hạn các kí hiệu trên băng

B : Một kí hiệu đặc biệt thuộc Γ gọi là kí hiệu trắng.

Σ : Tập con của Γ , không chứa B, được gọi là bộ chữ vào(kí hiệu kết thúc)

q_0: Trạng thái đầu

$F \subseteq Q$: Tập trạng thái kết thúc.

δ: Hàm chuyển trạng thái

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

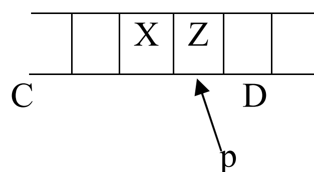
L, R là các trạng thái: trái, phải

Một hình trạng của máy Turing là một xâu có dạng #γ_1q γ_2#, trong đó # là một ký hiệu không thuộc Γ , # gọi là ký hiệu mút ; còn γ_1, γ_2 $\in\Gamma^*$, q $\in$Q. Hình trạng đầu là

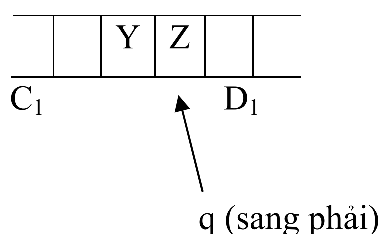
#q_0w # với w$\in\Sigma^*$

Ví dụ 1:

Thời điểm t



Thời điểm t+1



Hình 1: Một bước hoạt động của máy Turing

Tại thời điểm t máy Turing ở trạng thái p, đầu đọc /viết nhòm vào ô nhớ có kí hiệu là X. Tại thời điểm tiếp theo t+1 (một đơn vị thời gian) máy ở trạng thái q, ký hiệu X đã thay bằng Y, đầu đọc/viết chuyển sang trái hoặc sang phải.

δ: (p,X)→(q,Y,d) d∈{L,R}

hay viết pX→qYd gọi là một mệnh lệnh của máy T, xâu kí tự CpXD gọi là một hình trạng của máy T.

CpXD→C_1qZD_1 gọi là một bước chuyển hình trạng, nếu q∈F thì xem như quá trình xử lý kết thúc hay C_1qZD_1 là hình trạng cuối cùng.

- Nếu δ là hàm đơn trị thì T được gọi là máy tất định(đơn định)
- Nếu δ là hàm đa trị thì T được gọi là máy không tất định(không đơn định)
- Đơn vị nhớ: Là ô nhớ chứa một kí hiệu, nếu dùng mã nhị phân thì đơn vị nhớ là 1 bit.
- Đơn vị thời gian: Là thời gian để thực hiện một bước hoạt động cơ bản (bước chuyển hình trạng).

Nhận xét: Máy Turing có cấu tạo cực kì đơn giản nhưng lại làm được mọi việc liên quan tới tính toán các phép tính. Từ mô hình này có thể định nghĩa ra phép cộng (mã hóa dạng nhị phân) bằng cách dịch chuyển đầu đọc 0, 1 và từ đó định nghĩa ra các phép tính khác.

1.2.2. Máy xử lý thuật toán viết bằng ngôn ngữ tựa ALGOL

- Đơn vị nhớ: Một ô nhớ chứa trọn vẹn một dữ liệu.
- Đơn vị thời gian: Thời gian để thực hiện một phép tính cơ bản trong số học hay logic như cộng, trừ, nhân, chia, gán, so sánh…

1.3. Khái niệm thuật toán và độ phức tạp của thuật toán

1.3.1. Thuật toán(*Algorithm*)

Thuật toán được hiểu đơn giản là một dãy hữu hạn các qui tắc. Với cấu tạo và hoạt động của máy Turing, ta có thể định nghĩa một cách hình thức thuật toán chính là một máy Turing.

Ta đã có 2 mô hình tính toán là máy Turing và máy xử lý thuật toán viết bằng ngôn ngữ tựa ALGOL. Ứng với hai mô hình tính toán này có 2 cách biểu diễn thuật toán:

+ Thuật toán được biểu diễn bằng ngôn ngữ máy Turing.

+ Thuật toán được biểu diễn bằng ngôn ngữ tựa ALGOL.

1.3.2 Chi phí phải trả cho một quá trình tính toán và các khái niệm về độ phức tạp thuật toán

1.3.2.1. Chi phí phải trả cho một quá trình tính toán

Thường quan tâm tới chi phí thời gian và chi phí không gian (*bộ nhớ*)

- Chi phí thời gian của một quá trình tính toán là thời gian cần thiết để thực hiện một quá trình tính toán.

  + Với máy Turing: Chi phí thời gian là số bước chuyển hình trạng từ hình trạng đầu đến hình trạng kết thúc.

  + Với thuật toán tựa Algol: Chi phí thời gian là số các phép tính cơ bản cần thực hiện trong quá trình tính toán.

- Chi phí không gian của một quá trình tính toán là số ô nhớ cần để thực hiện một quá trình tính toán.

Gọi A là một thuật toán tương ứng với một mô hình tính toán

Gọi e là bộ dữ liệu vào đã được mã hóa theo cách nào đó

Khi đó thuật toán A tính trên dữ liệu e cần phải trả một giá nhất định bao gồm 2 giá:

+ $t_A(e)$ là giá thời gian

+ $l_A(e)$ là giá bộ nhớ

Cùng một thuật toán A, xử lý trên các bộ dữ liệu khác nhau thì sẽ có giá khác nhau.

Ví dụ 2: Cho dãy số nguyên $S=\{x_1, x_2, \dots x_n\}$, số phần tử n.

Tìm số lớn nhất của dãy ?

Bài toán được biểu diễn như sau.

Input: Dãy số nguyên $S=\{x_1, x_2, \ldots x_n\}$, n

Ouput: Số lớn nhất $Max=\max\{x_i\}$ của S.

Thuật toán A:

```
Begin      Max:=x1;
               For i:=2 to n do
                   If xi>Max then Max:=xi;
End.
```

* Xét bộ dữ liệu vào $e_1=\{4, 0, 9, 1, 5\}$

$l_A(e_1)=5+1+1+1=8$ (số biến vào:6, số biến ra:1, số biến phụ:1)

$t_A(e_1)=5+1=6$ vì

max:=4	thực hiện	1 phép tính
vì $x_2=0<max=4$ nên không làm gì	thực hiện	1 phép tính
$x_3=9>max=4$ nên max:=9	thực hiện	2 phép tính
$x_4=1<max=9$ nên không làm gì	thực hiện	1 phép tính
$x_5=5<max=9$ nên không làm gì	thực hiện	1 phép tính
⇒Tổng cộng thực hiện:		6 phép tính

* Xét bộ dữ liệu vào $e_2=\{2, 7, 8, 11, 17\}$ ta có:

$l_A(e_2)=8$

$t_A(e_2)=1+4.2 = 9$

Như vậy với $e_1 \neq e_2$ chi phí xử lý của A trên e_1 và e_2 là khác nhau.

b) Các khái niệm về độ phức tạp của thuật toán

✓ Độ phức tạp trong trường hợp xấu nhất

Cho một thuật toán A với đầu vào n, khi đó:

- Độ phức tạp về bộ nhớ trong trường hợp xấu nhất được định nghĩa là:

$$L_A(n) = \max\{l_A(e) \,\|\, |e| \le n\}$$

Tức là chi phí lớn nhất về bộ nhớ.

Trong ví dụ trên: Dữ liệu vào: n+1, ra:1, phụ:1 nên $L_A(e)=n+3$.

- Độ phức tạp thời gian trong trường hợp xấu nhất được định nghĩa là :

$$T_A(n) = \max\{t_A(e) \,\|\, |e| \le n\}$$

Tức là chi phí lớn nhất về thời gian.

Trong ví dụ trên $T_A(n) =1+2(n-1) = 2n-1$.

✓ **Độ phức tạp trung bình**

Là tổng số các độ phức tạp khác nhau ứng với các bộ dữ liệu chia cho tổng số.

✓ **Độ phức tạp tiệm cận**

Thuật toán A với đầu vào n gọi là có độ phức tạp O(f(n)) nếu ∃ hằng số C, N_0:

$T_A(n) \le C.f(n)$, $\forall n \ge N_0$. Tức là $T_A(n)$ có tốc độ tăng là O(f(n))

✓ **Độ phức tạp đa thức(Polynomial)**

Thuật toán được gọi là có độ phức tạp đa thức nếu tồn tại đa thức P(n) mà $T_A(n) \le C.P(n)$, $\forall n \ge N_0$.

✓ **Thuật toán đa thức**

Thuật toán được gọi là đa thức nếu độ phức tạp về thời gian trong trường hợp xấu nhất của nó là đa thức.

Việc đánh giá đúng độ phức tạp của bài toán là một vấn đề hết sức phức tạp. Vì vậy người ta thường quan tâm đến việc đánh giá độ phức tạp thời gian trong trường hợp xấu nhất của bài toán.

Một số đơn vị đo tốc độ tăng:

- O(1): Hầu hết các chỉ thị của chương trình đều được thực hiện một lần hay nhiều nhất chỉ một vài lần⇒Thời gian chạy của chương trình là hằng số.
- O(logN): Thời gian chạy của chương trình là logarit, tức là thời gian chạy của chương trình tiến chậm khi N lớn dần.
- O(N):Thời gian chạy là tuyến tính. Đây là tình huống tối ưu cho một thuật toán phải xử lý N dữ liệu nhập.
- O(NlogN): Thời gian chạy tăng dần lên cho các thuật toán mà giải một bài toán bằng cách tách nó thành các bài toán con nhỏ hơn, sau đó tổ hợp các lời giải.
- $O(N^2)$: Thời gian chạy là bậc 2, trường hợp này chỉ có ý nghĩa thực tế cho các bài toán tương đối nhỏ. Thời gian bình phương thường tăng dần trong các thuật toán phải xử lý tất cả các cặp phần tử dữ liệu (2 vòng lặp lồng nhau).
- $O(N^3)$: Thuật toán xử lý các bộ ba của các phần tử dữ liệu (3 vòng lặp lồng nhau)⇒ý nghĩa với các bài toán nhỏ.
- $O(2^n)$, O(n!), $O(n^n)$: Thời gian thực hiện thuật toán là rất lớn do tốc độ tăng của các hàm mũ.

1.4. Cách tính độ phức tạp

1.4.1. Các quy tắc cơ bản

a) Quy tắc cộng: *Nếu $T_1(n)$ và $T_2(n)$ là thời gian thực hiện 2 chương trình P_1, P_2 và $T_1(n)=O(f(n))$, $T_2(n)=O(g(n))$ thì thời gian thực hiện của đoạn 2 chương trình đó nối tiếp nhau là $T(n)=O(max(f(n),g(n))$*

Ví dụ: Lệnh gán x:=5 tốn một hằng thời gian $\approx$ O(1).

Lệnh đọc dữ liệu READ(x) tốn một hằng $\approx$ O(1).

Thời gian thực hiện cả 2 lệnh trên nối tiếp nhau là O(max(1,1))=O(1).

b) Quy tắc nhân: *Nếu $T_1(n)$ và $T_2(n)$ là thời gian thực hiện 2 đoạn chương trình P_1, P_2 và $T_1(n)=O(f(n))$, $T_2(n)=O(g(n))$ thì thời gian thực hiện của 2 đoạn chương trình đó lồng nhau là $T(n)=O(f(n).g(n))$.*

c) Quy tắc tổng quát để phân tích một chương trình

- Thời gian thực hiện của mỗi lệnh gán, READ, WRITE là O(1)

- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng quy tắc cộng$\Rightarrow$Thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.

- Thời gian thực hiên cấu trúc IF là thời gian lớn nhất thực hiện câu lệnh sau THEN hoặc ELSE và thời gian kiểm tra điều kiện, thường thời gian kiểm tra điều kiện là O(1).

- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích số lần lặp với thời gian thực hiện thân vòng lặp.

Ví dụ 3: Tính thời gian thực hiện đoạn chương trình:

Begin

1**. for** i:=1 **to** n-1 **do** *{lặp n-1 lần}.*

2**. for** j:=n **downto** i+1 **do** *{thực hiện (n-i)lần,mỗi lần O(1)$\Rightarrow$*

O((n-i).1)=O(n-i).

3**. if** a[j-1]>a[j] **then**

begin

đổi chỗ (a[i],a[j]).

4. temp:=a[j-1]; O(1)

5. a[j-1]:=a[i];

6. a[j]:=temp;

end.

End.

Độ phức tạp T(n)= $\sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$ =O(n^2).

Chú ý: Độ phức tạp thuật toán không chỉ phụ thuộc vào kích thước, thời gian mà còn phụ thuộc vào tính chất của dữ liệu vào.

Ví dụ 4: Thuật toán sắp xếp dãy số nguyên tăng dần. Nếu dãy nhập vào đã có thứ tự thì thời gian thực hiện khác với khi nhập vào dãy chưa có thứ tự

1.4.2. Độ phức tạp của các chương trình đệ quy

Với các chương trình chương trình đệ quy, trước hết ta cần thành lập các phương trình đệ quy, sau đó giải các phương trình đệ quy. Nghiệm của phương trình đệ quy là thời gian thực hiện chương trình đệ quy đó.

a)Thành lập phương trình đệ quy:

Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k) trong đó T(n) là thời gian thực hiện với kích thước dữ liệu nhập là n,

T(k) là thời gian thực hiện với kích thước dữ liệu nhập là k, k<n.

Để thành lập phương trình đệ quy ta căn cứ vào chương trình đệ quy.

Ví dụ 5: Hàm tính giai thừa viết bằng giải thuật đệ quy sau:

```
Function  Giai_thua(n:Integer):Integer;
Begin
        If n=0 then Giai_thua:=1
        Else Giai_thua:=n*Giai_thua(n-1);
End.
```

Gọi T(n) : Thời gian thực hiện tính n!

T(n-1) : Thời gian thực hiện tính (n-1)!

Trường hợp n = 0 ⟶ Thực hiện một lệnh gán Giai_thưa:=1 $\Rightarrow$ O(1) $\Rightarrow$ $T(0)=C_1$

Trường hợp n>0 $\Rightarrow$ Gọi đệ quy Giai_thua(n-1) tốn T(n-1) thời gian

Sau khi có kết quả của việc gọi đệ quy, phải nhân kết quả đó với n và gán cho Giai_thua, thời gian thực hiện phép nhân và phép gán là một hằng C_2.

Vậy ta có phương trình đệ quy là :

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

*Ví dụ 6: Xét thủ tục Mergesort sau:

```
Function  Mergesort(L:List;n:Integer):List;
Var  L1,L2:List;
  Begin
     If n=1 then return(L)
     Else
        Begin
            Chia L thành 2 nửa L1,L2 ,mỗi nửa có độ dài n/2
            Return(Merge(Mergesort(L1,n/2), Mergesort(L2,n/2));
        End;
  End;
```

Hàm Mergesort nhận một danh sách có độ dài n và trả về một danh sách đẫ được sắp xếp. Thủ tục Merge nhận 2 danh sách đã được sắp L_1, L_2 mỗi danh sách có độ dài n/2 trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự $\Rightarrow$ Thời gian thực hiện Merge các danh sách có độ dài n/2 là O(n).

- Gọi T(n) là thời gian thực hiện Mergesort 1 danh sách có n phần tử

  T(n/2) là thời gian thực hiện Mergesort 1 danh sách có n/2 phần tử

Ta có phương trình đệ quy :

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

Trong đó: - C_1 là thời gian phải tốn khi L có độ dài bằng 1

- Trường hợp n>1 , thời gian Mergesort được chia làm 2 phần:

+ Phần gọi đệ quy Mergesort 1 danh sách có độ dài n/2 là T(n/2)

+ Phần thứ 2 bao gồm phép thử n>1, chia danh sách thành 2 nửa và Merge, ba thao tác này có thời gian không đổi $\Rightarrow$ Thời gian thực hiện là C_2n

b. Giải phương trình đệ quy:

Phương pháp truy hồi:

Dùng đệ quy để thay thế bất kì T(m) với m<n vào phía phải của phương trình cho đến khi tất cả T(m) với m>1 được thay thế bởi biểu thức của T(1). Vì T(1) luôn là hằng nên ta có công thức của T(n) chứa các số hạng chỉ liên quan tới n và các hằng số.

*Ví dụ 7: Giải phương trình:

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

Ta có $T(n) = 2T\left(\frac{n}{2}\right) + C_2 n$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + C_2\frac{n}{2}\right] + C_2 n = 4T\left(\frac{n}{4}\right) + 2C_2 n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + C_2\frac{n}{4}\right] + 2C_2 n = 8T\left(\frac{n}{8}\right) + 3C_2 n$$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + iC_2 n$$

Giả sử $n=2^k$ quá trình suy rộng này sẽ kết thúc khi i=k $\Rightarrow T(n) = 2^k T(1) + kC_2 n$

Vì $2^k=n \Rightarrow k=\log n$ và với $T(1) = C_1 \Rightarrow T(n) = C_1 n + C_2 n \log n$)

Vậy thời gian thực hiện thuật toán là O(nlogn)

Định lý: (Về nghiệm của phương trình truy hồi)

Cho a, b, c nguyên, dương. Khi đó nghiệm của phương trình truy hồi:

$$T(n) = \begin{cases} b & \text{nếu } n = 1 \\ aT(\frac{n}{c}) + bn & \text{nếu } n > 1, n = c^k \end{cases}$$

Có dạng:

$$T(n) = \begin{cases} O(n) & \text{nếu } a < c \\ O(n\log_c n) & \text{nếu } a = c \\ O(n^{\log_c a}) & \text{nếu } a > c \end{cases}$$

1.5. Thuật toán không đơn định đa thức(Nodeterministic Polynomial NP)

Với nhiều bài toán tối ưu tổ hợp vẫn chưa tìm được các *thuật toán đơn định* chạy trong thời gian đa thức, trong khi đó nếu cho phép dùng *thuật toán không đơn định* thì lại dễ dàng chỉ ra các thuật toán chạy trong thời gian đa thức. Ta xét bài toán sau đây:

Bài toán xếp balô 0-1(KNASPACK)

- Input: 1 balô có thể tích B; n đồ vật có thể tích $a_1, a_2, \ldots, a_n$.
- Output: Tìm nhóm đồ vật đặt vừa khít balô.

*Cách 1: Phương pháp Vét toàn bộ cần số phép thử các khả năng là: $C_n^1 + C_n^2 + \ldots C_n^{n-1} + C_n^n \approx 2^n$ Độ phức tạp tính toán là $O(2^n)$.

*Cách 2: Diễn tả thuật toán không đơn định ta cần dùng 3 hàm:

- CHOICE($a_1, a_2, \ldots a_n$): Chọn một trong số n giá trị.
- SUCCESS: Nếu có một điều kiện thỏa mãn.
- FAILURE: Nếu điều kiện không thỏa mãn.

Khi đó bài toán trên có thể diễn đạt như sau:

Liệu có thể tồn tại tập chỉ số $T \subset \{1,2,\ldots,n\}$ mà $\sum_{i \in T} ai = B$.

Thuật toán:

For i:=1 **to** n **do**

x_i:= CHOICE({0,1}); *{phép toán lựa chọn một trong 2 giá trị}*

if $\sum_{i=1}^{n} x_i a_i$ =B **then** SUCCESS

else FAILURE;

- Giá phải trả về thời gian :

+ Trường hợp SUCCESS: Thời gian ít nhất để thực hiện SUCCESS .

+ Trường hợp FAILURE: Chính là thời gian tối đa.

Thuật toán trên trở thành không đơn định đa thức, số phép tính thực hiện là 2*n+2.

Bài toán Xếp balô mở rộng (Tên trộm tham lam)

Input: Một ba lô có thể tích B, n đồ vật có thể tích: $a_1, a_2,\ldots a_n$,

giá trị tương ứng của các đồ vật là: $p_1, p_2,\ldots p_n$

Ouput: Có tồn tại tập $T \subset \{1,2,\ldots,n\}$ sao cho $\sum_{i \in T} ai \le b$ và $\sum_{i \in T} p_i$ đạt max ?.

Bài toán xếp balô giá trị nguyên:

Input: Một ba lô có thể tích B, n đồ vật có thể tích: $a_1, a_2,\ldots a_n$,

giá trị tương ứng của các đồ vật là: $p_1, p_2,\ldots p_n$

Số lượng mỗi loại đồ vật là không hạn chế, x_i nguyên là số lượng loại đồ vật i.

Ouput: Tìm nhóm đồ vật thoả mãn $\sum_{i=1}^{n} a_i x_i \le B$ và $\sum_{i=1}^{n} p_i x_i$ đạt max ?.

✓ **Mối quan hệ về tính đa thức giữa mô hình Turing và mô hình tựa Algol**

Định lí 1: *Thuật toán trên máy Turing là đa thức thì thuật toán trên tựa Algol tương ứng cũng là đa thức, ngược lại chưa chắc.*

Ví dụ 8: Tính $S=2^{2^n}$.

```
x:=2;
for i:=1 to n do x:=x*x;
```

Ta có i:=1 : x^2

i:=2 : $x^2 * x^2 = x^{2^2}$

i:=3 : $x^4 * x^4 = x^{2^3}$

…

i:=n : $x^{2^{n-1}} * x^{2^{n-1}} = x^{2^n}$.

+ Trên máy Turing : Dữ liệu vào 2^{2^n} mã nhị phân là: $[\log_2 2^{2^n}] + 1 \approx 2^n$, độ phức tạp là $O(2^n)$

+ Thuật toán tựa Algol : Độ phức tạp $2n+1 \approx O(n)$.

Định lí 2 : *Nếu thuật toán tựa Algol là đa thức và trong thuật toán chỉ có các phép toán cơ bản(+, -, *, /, so sánh,gán, AND, OR…) và dữ liệu vào phải có độ phức tạp*

đa thức theo quan niệm 2(độ dài mã) thì thuật toán (trên máy Turing) tương ứng là đa thức.

Ví dụ: Input: Dãy số $a_1,a_2,\ldots a_n$, n.

Output: Sắp xếp theo chiều giảm dần.

```
For i:=1 to n do
Begin
    j:=i;
    for k:=i+1 to n do
        if a_k>a_j then j:=k;
    TAM:=a_i; a_i:=a_j; a_j:=TAM;
End;
```

Độ phức tạp tính toán:

- Dữ liệu: $n+1 \approx O(n)$.
- Bộ nhớ: $(n+1)+4=n+5 \approx O(n)$

(vào) (i,j,k,tam)

- Thời gian: $2((n-1)+(n-2)+\ldots+2+1)+4(n-1) = 2n.\frac{n-1}{2}+4(n-1) = n^2+3n-4 \approx O(n^2)$.

$\Rightarrow$ Thuật toán là đa thức $\Rightarrow$ Thực tế giải được.

1.5.1. Sự phân lớp các bài toán.

Với một bài toán cho trước có 2 khả năng xảy ra:

+ Không giải được hoặc

+ Giải được bằng thuật toán.

- Trường hợp bài toán giải được bằng thuật toán cũng chia làm 2 loại:

+ Thực tế giải được: Được hiểu là thuật toán xử lý trong thời gian đủ nhanh, thực tế cho phép, đó là thuật toán có độ phức tạp thời gian là đa thức.

+Thực tế khó giải: Được hiểu là thuật toán xử lý trong nhiều thời gian, thực tế khó chấp nhận, đó là thuật toán có độ phức tạp thời gian là trên đa thức (hàm mũ).

Do đó, ta có sự phân lớp các bài toán do 2 tác giả Cook và Karp đề xuất năm 1970-1971 như sau:

- **P** : Là lớp các bài toán có thể giải được bằng thuật toán đơn định trong thời gian đa thức (*Deterministic polynomial*).

Ví dụ: Bài toán về tính liên thông của đồ thị có thể giải được nhờ thuật toán với thời gian tính là $O(n^2) \Rightarrow$ Thuộc lớp P

- **NP** : Là lớp các bài toán có thể giải được bằng thuật toán không đơn định trong thời gian đa thức. Hay, là lớp các bài toán mà mọi nghiệm giả định đều có thể được kiểm chứng trong thời gian đa thức (*Nondeterministic polynomial*)..

Ví dụ: Bài toán kiểm tra một dãy đỉnh của đồ thị G có là chu trình Hamilton hay không có thể thực hiện sau thời gian đa thức ⇒ Thuộc lớp NP

$\Rightarrow \quad P \subseteq NP$

Nhưng hiện nay chưa chứng minh được P là tập con thực sự của NP, vấn đề P = NP? hiện là một trong số các vấn đề mở nổi tiếng nhất và cũng đắt giá nhất trong Toán học và trong Tin học lý thuyết.

1.5.2. Khái niệm "dẫn về được" (*Phép quy dẫn*): Cho hai bài toán A, B

Định nghĩa1: *Bài toán A được gọi là "dẫn về được" bài toán B sau thời gian đa thức nếu có một thuật toán đa thức để giải bài toán B thì cũng có một thuật toán đa thức để giải bài toán A.*

Nghĩa là: Bài toán B "khó hơn" bài toán A hay A "dễ hơn" B hay A là trường hợp riêng của B. Kí hiệu $A \propto B$.

Phép quy dẫn có tính chất bắc cầu: $A \propto B$ và $B \propto C \Rightarrow A \propto C$.

Tư tưởng quy dẫn đã giải thích vai trò quan trọng của lớp bài toán P. Nếu ta có bài toán A thuộc lớp P và một bài toán B có thể quy dẫn về A, thế thì B cũng thuộc vào P. Nghĩa là P là đóng đối với phép quy dẫn.

Định nghĩa 2 : *Bài toán A được gọi là "khó tương đương" bài toán B nếu $A \propto B$ và $B \propto A$. Kí hiệu $A \sim B$.*

1.5.3 Lớp bài toán NP - khó (*NP - hard*) và NP - đầy đủ (*NP – Complate*)

a) Bài toán quyết định: *Bài toán quyết định là bài toán mà đầu ra chỉ có thể là "Yes" hoặc "No" (Đúng/sai, 0/1, chấp nhận/từ chối).*

Ví dụ: Bài toán về tính nguyên tố: " Hỏi số nguyên n có là số nguyên tố hay không?". Khi đó ta có n = 23 là bộ dữ liệu vào "Yes", còn n = 24 là bộ dữ liệu vào " No" của bài toán.

b) Bài toán NP – Khó(NP – Hard)

Bài toán A được gọi là NP- khó nếu như tồn tại thuật toán đa thức để giải bài toán A thì kéo theo sự tồn tại thuật toán đa thức để giải mọi bài toán trong NP.

Hay: A là NP – Khó nếu như $B \propto A$, với mọi bài toán $B \in NP$

Một cách không hình thức, có thể nói rằng nếu ta có thể giải được một cách hiệu quả một bài toán NP – Khó cụ thể thì ta cũng có thể giải hiệu quả bất kỳ bài toán nào trong NP bằng cách sử dụng thuật toán giải bài toán NP-Khó như là một chương trình con.

c) Bài toán NP - đầy đủ(*NP – complete, NPC*)

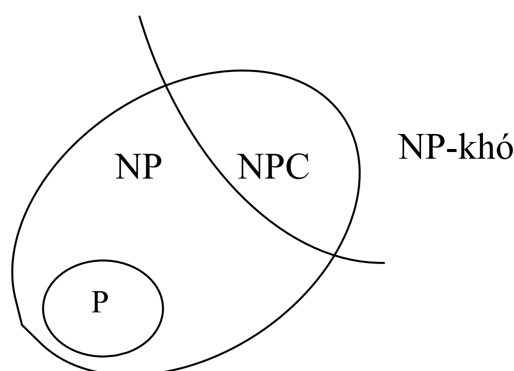
Một bài toán quyết định A được gọi là NP - đầy đủ nếu như

i) A là bài toán trong NP,

ii) Mọi bài toán trong NP đều có thể quy dẫn về A.

Lưu ý: Khái niệm NP - đầy đủ đòi hỏi bài toán nhất thiết phải có dạng quyết định.

Ta có bức tranh tạm thời đầy đủ về phân lớp các bài toán trên hình sau:



Hình 2: Sự phân lớp các bài toán

c) Phương pháp chứng minh một bài toán là NP - đầy đủ

- Cách 1: Theo định nghĩa (rất khó).
- Cách 2: áp dụng bổ đề sau:

Bổ đề: Giả sử bài toán A là NP - đầy đủ, bài toán B thuộc NP và bài toán A quy dẫn về B. Khi đó bài toán B cũng là NP - đầy đủ.

Ví dụ 9: Bài toán xếp balô(KNASPACK)$\in$ NPC$\Rightarrow$Chứng minh bài toán lập lịch $\in$NPC.

°Bài toán lập lịch (*Bài toán PHẠT*):

Input: Có n công việc xử lý trên một máy.

r_i: Thời điểm bắt đầu công việc xử lý i

d_i : Hạn định hoàn thành công việc i.

t_i : Thời gian xử lý công việc i, $t_i \le d_i - r_i$.

b_i : Thời gian bắt đầu xử lý.

c_i : Thời gian kết thúc công việc i, $t_i = c_i - b_i$

nếu $c_i \le d_i$, công việc i là xử lý đúng hạn.

nếu $c_i>d_i$, công việc i là xử lý quá hạn(bị phạt).

w_i : Tiền phạt.

Output: Hãy sắp xếp các công việc theo một thứ tự nhất định để theo đó chờ đến lượt xử lý, sao cho lượng tiền phạt là ít nhất.

Kí hiệu $U_i = \begin{cases} 0 \text{ nếu } c_i \leq d_i \text{ (đúng hạn)} \\ 1 \text{ nếu } c_i > d_i \text{ (quá hạn)} \end{cases}$

Khi đó yêu cầu : $\sum_{i=1}^{n} U_i W_i \rightarrow \min$

Ta có thể viết bài toán trên ngắn gọn như sau : $n|1|\sum_{i=1}^{n} U_i W_i$. Kí hiệu là PHẠT.

Bài toán này rõ ràng là giải được bằng phương pháp "vét toàn bộ". Nhưng thực tế khó giải vì nó thuộc lớp NP_đầy đủ.

Để chứng minh bài toán "PHẠT" là NP - đầy đủ, chỉ cần chứng minh rằng bài toán KNAPSACK $\propto$ PHẠT vì ta đã biết KNAPSACK là NP_đầy đủ. Nói một cách khác KNAPSACK là trường hợp riêng của PHẠT.

Nhắc lại bài toán KNAPSACK:

Input: n đồ vật với thể tích $a_1, a_2, \ldots a_n$ cần nhét vào balô có thể tích B.

Output: Tìm nhóm đồ vật có thể nhét vừa khít balô trên.

($\exists T \subseteq \{1,2,\ldots,n\}$ mà $\sum_{i \in T} ai = B$.)

a) Để chứng minh KNAPSACK $\propto$ PHẠT trước hết ta diễn đạt nó bằng ngôn ngữ của bài toán PHẠT. Cụ thể mỗi vật i ở KNAPSACK được xem là một công việc trong PHẠT, chúng đồng thời được nhập vào hệ thống. Mọi công việc có hạn định như nhau và bằng B. Thời gian t_i thực hiện công việc i bằng tiền phạt w_i và bằng thể tích a_i của vật.

Tóm lại ta có thể biểu diễn bài toán như sau:

Input:
- n công việc đồng thời được xử lý $r_i = 0$, $\forall$ i=1,2,…,n.
- mỗi công việc i ($1 \leq i \leq n$) được biết d_i=B, $t_i=w_i=a_i$, $\forall$ i=1,2,…,n.
- máy làm việc liên tục cho đến khi mọi công việc được xử lý xong.
- tại mỗi thời điểm máy chỉ xử lý được một công việc.
- khi đang xử lý công việc i, không được phép ngắt nó để thực hiện một công việc khác.

Output: Hãy lập lịch để máy xử lý các công việc sao cho lượng tiền phạt là ít nhất

$\sum_{i=1}^{n} U_iW_i$ là nhỏ nhất.

b) Chứng minh:

Giải được PHẠT bằng thuật toán đơn định đa thức thì cũng giải được KNAPSACK bằng thuật toán đơn định đa thức và ngược lại.

°Giả sử giải được PHẠT tức là lịch biểu mà $\sum_{i=1}^{n} W_iU_i$ là nhỏ nhất, vậy thì:

$\sum_{i=1}^{n} W_iU_i=\sum_{i=1}^{n} a_i$ -b.

$$\sum b_i = \sum a_i = \sum w_i$$

$r_i=0$ b $d_i=b$ $\sum a_i$ -b

Suy ra $\exists S \subseteq \{1,2,\dots,n\}$ mà $\sum_{i=1}^{n} U_iW_i = \sum_{i\in S} a_i = \sum_{i=1}^{n} a_i$ -b

Hay b=$\sum_{i=1}^{n} a_i - \sum_{i\in S} a_i = \sum_{i\in T} a_i$ với T={1,2,…,n}- S. Như vậy KNAPSACK đã giải được.

°Ngược lại, giả sử KNAPSACK đã giải được, tức là $\exists T \subseteq \{1,2,\dots,n\}$ mà $\sum_{i\in T} a_i = b$

hay

$\sum_{i\in T} a_i = \sum_{i=1}^{n} a_i - \sum_{i\in S} a_i$ =b, như vậy $\sum_{i=1}^{n} a_i - \sum_{i=1}^{n} U_iW_i$ =b$\Rightarrow$ $\sum_{i=1}^{n} U_iW_i = \sum_{i=1}^{n} a_i$ -b , đây là lượng tiền nhỏ nhất và PHẠT đã giải được.

*Chú ý: Nếu tất cả n công việc đều quá hạn thì lượng tiền phạt lớn nhất là $\sum_{i=1}^{n} a_i$.

d) Một số bài toán đã được chứng minh là NP – khó , NP - đầy đủ

Để chứng minh một bài toán nào đó là NP-đầy đủ (NP-khó) công việc khó khăn nhất là tìm được một bài toán NP-đầy đủ có thể quy dẫn về nó. Do đó ta cần biết thêm về những bài toán đã được chứng minh là NP-đầy đủ, cho đến nay danh mục các bài toán NPC trong các lĩnh vực đa dạng :Logic Bool, đồ thị, số học, lập lịch, trò chơi, otomat…đã lên đến hàng nghìn . Sau đây là một số bài toán đã được chứng minh là NPC:

① Bµi to¸n 3-SAT.

Xét các biểu thức Bool là hội của các mệnh đề mà mỗi mệnh đề là tuyển của đúng 3 toán hạng, mỗi toán hạng là một biến Bool (x) hoặc phủ định của nó $(\overline{x})$. Biểu thức Bool có dạng như vậy được gọi là công thức 3-CNF *(dạng chuẩn tắc hội 3 – conjunctive normal form).*

Ví dụ. Biểu thức $(x \vee y \vee z) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee z \vee u)$

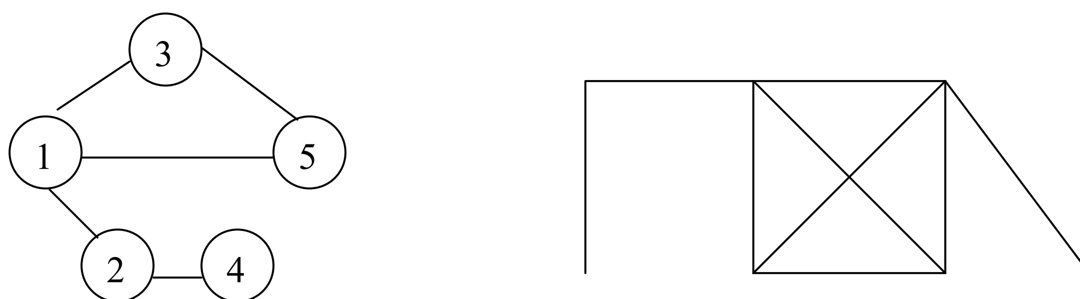
Là một 3-CNF chứa 4 biến Bun x, y, z, u.

Bài toán 3-SAT: Cho một công thức 3-CNF, hỏi rằng có tồn tại một bộ giá trị của các biến sao cho biểu thức nhận giá trị TRUE hay không?

② Bài toán về bè lớn nhất của đồ thị (MaxClique):

Cho đồ thị vô hướng G = (V, E). Một đồ thị con đầy đủ của đồ thị G được gọi là bè (clique). Ta gọi kích thước của bè là số đỉnh của nó. Bè của đồ thị G với kích thước lớn nhất được gọi là bè lớn nhất(MaxClique)

Ví dụ :



Hình 3 :

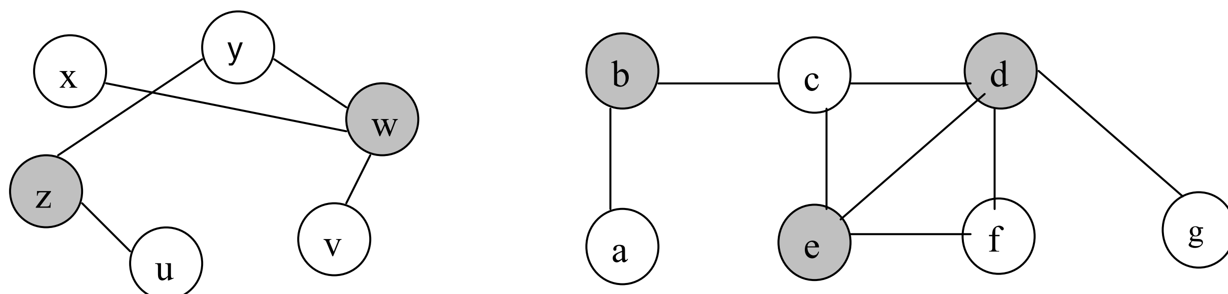
a)*MaxClique kích thước 3* b)*MaxClique kích thước 4*

Bài toán Clique: Cho đồ thị vô hướng G = (V, E) và số nguyên k. Hỏi đồ thị G có chứa bè với kích thước hay không *?*

③.Bài toán phủ đỉnh(Vertex Cover- VC) : Ta gọi một phủ đỉnh của đồ thị vô hướng G=(V, E) là một tập con các đỉnh của đồ thị S $\subseteq$ V sao cho mỗi cạnh của đồ thị có ít nhất một đầu mút trong S. Ta gọi kích thước của một phủ đỉnh là số đỉnh của nó

Bài toán VC : Cho đồ thị vô hướng G=(V, E) và số nguyên k. Hỏi có phủ đỉnh với kích thước k hay không?

Ví dụ :



Hình 4

a*) Phủ đỉnh với kích thước 2* *b) Phủ đỉnh với kích thước 3*

1.6. Thuật toán xấp xỉ (*Heuristic*)

1.6.1. Các khái niệm

Người ta cho rằng ngày nay máy tính với tốc độ rất lớn, không cần quan tâm nhiều tới thuật toán nhanh nhưng với sự kiểm chứng sau đây: Bài toán xử lý với n đối tượng, có 3 thuật toán với 3 mức phức tạp khác nhau, sau 1 giờ xử lý sẽ chịu 3 hậu quả khác nhau.

Thuật toán	Độ phức tạp	Xử lý/1 giờ
A	$O(n)$	3,6 triệu đối tượng
B	$O(n\log_2 n)$	0,2 triệu đối tượng
C	$O(2^n)$	21 đối tượng

Trong khi đó nhiều bài toán có ý nghĩa thực tế lại thuộc lớp các bài toán NPC và rất quan trọng. Nếu một bài toán là NPC ta ắt không tìm một thuật toán thời gian đa thức . Vì vậy, có hai cách tiếp cận để có thể khắc phục tính NPC:

- Nếu dữ liệu đầu vào thực tế là nhỏ thì một thuật toán có thời gian thực hiện hàm mũ có thể hoàn toàn thoả mãn.
- Tìm các giải pháp gần tối ưu trong thời gian đa thức.

Một thuật toán trả về các kết qủa gần tối ưu được gọi là một thuật toán xấp xỉ.

Ta có các khái niệm sau đây:

- *Thuật toán tối ưu nhanh*: Là thuật toán tìm nghiệm tối ưu, nhưng nhanh (độ phức tạp thời gian là đa thức).
- *Thuật toán tối ưu chậm*: Là thuật toán tìm nghiệm tối ưu nhưng chậm (độ phức tạp thời gian là hàm mũ).

• *Thuật toán xấp xỉ nhanh* (*Fasf Approximation Algorithms*). Là các thuật toán tìm ra nghiệm gần đúng của bài toán với độ chính xác nào đó nhưng đủ nhanh. Thuật toán như vậy còn được gọi là "Thuật toán xấp xỉ đa thức".

Ví dụ 10: *Bài toán phủ đỉnh tối ưu*

Input: Cho đồ thị vô hướng G = (V, E).

Output: Tìm phủ đỉnh tối ưu(Phủ đỉnh có kích thước cực tiểu) .

Bài toán VC tìm ra phủ đỉnh có kích cỡ cực tiểu là NPC. Do đó khó có thể tìm ra 1 phủ đỉnh tối ưu nhưng không quá khó để tìm ra một phủ đỉnh gần tối ưu.

Sau đây là một thuật toán xấp xỉ cho kết qủa là một phủ đỉnh có kích cỡ không lớn hơn 2 lần kích cỡ một phủ đỉnh tối ưu trong thời gian đa thức:

Procedure Approx _VertexCover;

Begin

C:= ϕ; { C - tập phủ gần tối ưu}

E:= Tập cạnh của đồ thị G;

While E ≠ ϕ **do**

Begin

Chọn (u, v) là một cạnh tuỳ ý của E;

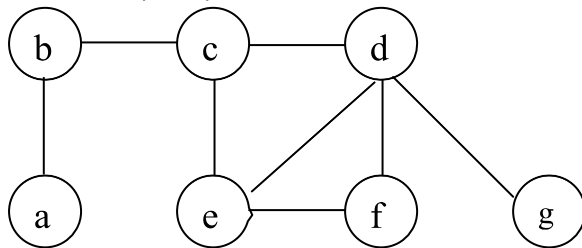
C:= C ∪ {u, v}; {Kết nạp hai đỉnh u, v vào phủ đỉnh C};

Gỡ bỏ khỏi E mọi cạnh liên thuộc với u hoặc v;

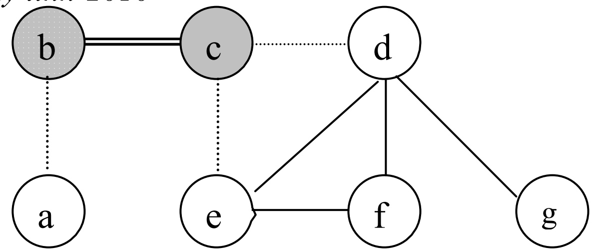
End;

Return(C);

End;

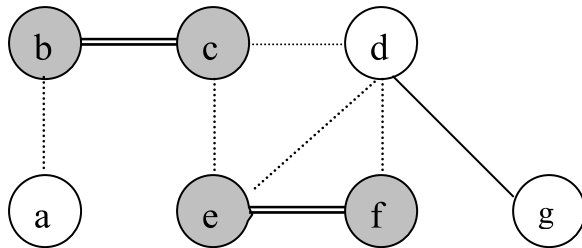


a) Đồ thị G

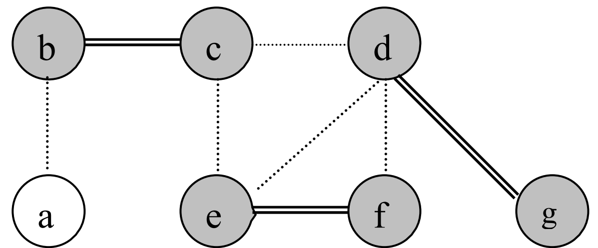


b) Chọn cạnh (b, c), gỡ bỏ (b,a),(c,e),(c,d)

Hình 5



c) Chọn tiếp (e,f), gỡ bỏ (e,d),(d,f)



d) Chọn cạnh (d,g), E = ϕ

Kết quả : Phủ đỉnh C={b, c, d, e, f, g} gần tối ưu có kích thước 6.

(Phủ đỉnh tối ưu {b, e, d} có kích thước 3)

1.6.2. Thuật toán ε - xấp xỉ tuyệt đối

Cho P là bài toán cực đại hóa:

Gọi H là thủ tục Heuristic, thuật toán tìm một nghiệm nào đó cho P

Kí hiệu OPT(I) là nghiệm tối ưu của bài toán P đối với thể hiện I.

Kí hiệu H(I) là nghiệm gần đúng của P do thuật toán H tìm ra.

Cho ε>0, thủ tục Heuristic H được gọi là thuật toán ε - xấp xỉ tuyệt đối khi và chỉ khi $|OPT(I) - H(I)| \le \varepsilon$ cho mọi thể hiện I của bài toán P (I: instance) với $\forall$ bộ dữ liệu.

Ví dụ 11: *Bài toán lưu trữ tối đa số lượng chương trình (maximum program stored)*

Input : - n chương trình với dung lượng nhớ (độ dài) $d_1,d_2,\ldots, d_n$

- Hai băng nhớ với dung lương (độ dài) mỗi băng là L.

Output: Hãy ghi các chương trình lên 2 băng nhớ với số lượng tối đa, mỗi chương trình chỉ được ghi trên một băng nhớ.

Bài toán này đã được chứng minh là NP - đầy đủ. Vì vậy viêc tìm thuật toán đa thức cho nó là ít hi vọng.

Người ta đã dùng giải pháp tìm thuật toán xấp xỉ nhanh cho phép tìm được nghiệm gần đúng của nó nhưng chỉ mất thời gian đa thức.

Sau đây là thuật toán 1- xấp xỉ tuyệt đối: Cho kết quả nghiệm tối ưu và nghiệm gần đúng chỉ chênh nhau có 1.

Thuật toán 1- xấp xỉ tuyệt đối:

```
Procedure  XXTD1;
  Begin
    1. Sắp xếp các chương trình theo thứ tự tăng dần của d1,d2,…, dn;
    2. i:=1;
      For j:=1 to 2 do
        Begin
          dodai:=0;
          While (dodai+di ≤L) and (i≤n) do
            Begin
              <ghi chương trình i vào băng nhớ j>;
              dodai:= dodai+di;
              i:=i+1;
            End;
        End;
  End;
```

Chú ý: Mục đích muốn chỉ trên 2 băng nhớ có độ dài L mà lưu trữ được tối đa các chương trình. Theo suy nghĩ thông thường thì hãy ưu tiên các chương trình có độ dài ngắn hơn, vì vậy đầu tiên là sắp xếp các chương trình theo thứ tự tăng dần các độ dài của chúng. Tiếp theo lần lượt xếp theo thứ tự này lên từng băng nhớ một. Do 2 băng nhớ có độ dài như nhau nên dùng băng nào trước cũng được . Theo thuật toán trên thì dùng băng 1 trước. Biến "dodai" ghi lại tổng độ dài băng nhớ dùng để lưu các chương trình.

Bài toán này còn được gọi là bài toán cắt n đoạn sắt từ 2 thanh sắt có cùng độ dài L sao cho số lượng đoạn sắt cắt ra là nhiều nhất (Chặt sắt - Cutting problem).

Chứng minh $|OPT(I) - H(I)| \leq 1$

- Đặt k = H(I) là nghiệm của thuật toán Heurtstic $\Rightarrow$ k là số lượng chương trình được lưu trữ trên 2 băng nhớ theo cách sắp đặt của thuật toán xấp xỉ trên.

- Gọi p là số lượng chương trình được ghi trên 1 băng nhớ có độ dài bằng 2 băng nói trên

Như vậy $k \le OPT(I) \le p$ và $\sum_{i=1}^{p} d_i \le 2L$ **(1)**

Ta chứng minh $|OPT(I) - H(I)| \le 1 \Leftrightarrow OPT(I) - k \le 1 \Leftrightarrow OPT(I) \le k+1$ **(2)**

Theo **(1)** thì chỉ cần chứng minh $p \le k+1$ là đủ vì khi đó $OPT(I) \le p \le k+1$.

<u>Chứng minh bằng phản chứng:</u>

Giả sử $p>k+1 \Leftrightarrow p \ge k+2 \Leftrightarrow \sum_{i=1}^{k+2} d_i \le \sum_{i=1}^{p} d_i \le 2L$ **(3).**

Gọi m là số lượng chương trình được ghi trên một băng theo thuật toán xấp xỉ trên.

Khi đó : $\sum_{i=1}^{m} d_i + d_{m+1} > L \Rightarrow \sum_{i=1}^{m} d_i + d_{k+1} > L$ **(4)**

Tương tự trên băng nhớ 2 : $\sum_{i=m+1}^{k} d_i + d_{k+1} > L \Rightarrow \sum_{i=m+1}^{k} d_i + d_{k+2} > L$ **(5)**

Từ **(4),(5)** ta có : $\sum_{i=1}^{k+2} d_i > 2L \Rightarrow$ mâu thuẫn với (3) $\Rightarrow p \le k+1$ (đpcm).

Độ phức tạp thời gian của thuật toán:

Thời gian xử lý của thuật toán xấp xỉ trên là $O(n\log_2 n)$ (chủ yếu là phần sắp xếp các chương trình theo thứ tự của độ dài). Trong khi thuật toán chính xác phải cần có thời gian hàm mũ, mà hiệu quả là 2 nghiệm chỉ chênh nhau có 1. Nếu những bài toán được giải tốt như ví dụ trên thì dùng giải pháp ε- xấp xỉ tuyệt đối. Nhưng không phải khi nào cũng suôn sẻ như vậy vì các thuật toán ε- xấp xỉ tuyệt đối tìm được không nhiều.

Hiện nay phần lớn các bài toán NP- đầy đủ thì việc tìm thuật toán ε- xấp xỉ tuyệt đối cho chúng cũng lại là NP- đầy đủ. Chẳng hạn như bài toán xếp balô (KNAPSACK), bài toán người bán hàng (Traverling Salesman Problem), bài toán MaxClique…Chính vì lẽ đó người ta dẫn ra khái niệm yếu hơn gọi là Thuật toán ε- xấp xỉ.

1.6.3.Thuật toán ε- xấp xỉ

Cho P là bài toán cực đại hóa.

Gọi H là thủ tục Heuristic, thuật toán xấp xỉ tìm một nghiệm nào đó cho P.

Kí hiệu OPT(I) là nghiệm tối ưu của bài toán P đối với thể hiện I (Instance).

H(I) là nghiệm gần đúng của P do H tìm ra.

Thủ tục Heuristic H được gọi là thuật toán ε - xấp xỉ khi và chỉ khi:

$$\frac{OPT(I)-H(I)}{OPT(I)} \le \varepsilon \text{ cho } \forall \text{I.}$$

Ví dụ 12: *Bài toán xếp balô giá trị nguyên (Interger - Valued Knapsack)*

Input: Một ba lô có thể tích B, n đồ vật có thể tích: $a_1, a_2,\dots a_n$,

giá trị tương ứng của các đồ vật là: $p_1, p_2,\dots p_n$

Số lượng mỗi loại đồ vật là không hạn chế, x_i nguyên là số lượng loại đồ vật i.

Ouput: Tìm nhóm đồ vật thoả mãn $\sum_{i=1}^{n} a_i x_i \le B$ và $\sum_{i=1}^{n} p_i x_i$ đạt max ?.

Tóm tắt: $\sum_{i=1}^{n} x_i p_i \to$ Max với $\sum_{i=1}^{n} x_i a_i \le B, x_i \in Z, 0 \le x_i \le b_i$. Ở đây $b_i \le \left[\frac{B}{w_i}\right]$, điều này là hiển nhiên vì $\left[\frac{B}{w_i}\right]$ chính là số nguyên đồ vật có cùng thể tích a_i có thể nhét được vào ba lô.

Trường hợp b_i=1 $\forall$ i thì vấn đề trên gọi là bài toán xếp balô 0-1, tức là chỉ được xếp nhiều nhất là 1 đồ vật vào balô (0-1 Knapsack)

Bài toán này đã được chứng minh là NP- đầy đủ. Vì vậy việc tìm thuật toán đa thức cho nó là rất ít hi vọng. Người ta đã thử tìm thuật toán xấp xỉ tuyệt đối (nhanh) cho nó nhưng cũng không thành công vì việc tìm một thuật toán như vậy cũng lại là NP- khó.

Sau đây là thuật toán 1/2 - xấp xỉ cho bài toán xếp balô trị nguyên:

```
Procedure  XAPXI12;
  Begin
      1) sắp xếp tỉ số pi/ai , i=1,2,..,n theo thứ tự giảm dần;
      2) T:=0;
         for i:=1 to n do
         begin
             xi:=[(B-T)/ai];
             T:=T+xiai;
         End;
  End.
```

Chứng minh:

- Với mỗi thể hiện I ta có $OPT(I) \le p_1.\frac{B}{a_1}$ và $p_1.\left\lfloor \frac{B}{a_1} \right\rfloor \le H(I)$

Mặt khác $B - \left\lfloor \frac{B}{a_1} \right\rfloor .a_1 < \frac{B}{2}$ (vì nếu ngược lại thì dẫn đến vô lý)

Khi đó $\frac{OPT(I) - H(I)}{OPT(I)} = 1 - \frac{H(I)}{OPT(I)} \le 1 - \frac{\left\lfloor \frac{B}{a_1} \right\rfloor}{\frac{B}{a_1}} = \frac{B - a_1\left\lfloor \frac{B}{a_1} \right\rfloor}{B} = \frac{B/2}{B} = \frac{1}{2}$.

Độ phức tạp thời gian của thuật toán:

Thời gian xử lý thuật toán xấp xỉ trên chỉ là O(nlogn) (chủ yếu là phần sắp xếp tỉ số p_i/a_i), trong khi nếu dùng thuật toán chính xác phải cần thời gian hàm mũ.

Ngoài bài toán xếp balô (Knapsack) trên, hiện nay người ta đã tìm được thuật toán ε-xấp xỉ cho nhiều bài toán khác, đặc biệt trong các vấn đề lập lịch.

Tuy vậy với nhiều bài toán NP- đầy đủ thì việc tìm thuật toán ε- xấp xỉ cho chúng cũng lại là NP- đầy đủ. Chẳng hạn như bài toán Traveling Salesman Problem(TSP), bài toán quy hoạch nguyên (Integer programming)..

CHƯƠNG 2

CÁC THUẬT TOÁN SẮP XẾP

2.1. Bài toán sắp xếp

2.1.1. Tầm quan trọng của bài toán sắp xếp

Sắp xếp một danh sách các đối tượng theo một thứ tự nào đó là một bài toán thường được vận dụng trong các ứng dụng tin học, và là một yêu cầu không thể thiếu trong khi thiết kế các phần mềm.

2.1.2. Sắp xếp trong và sắp xếp ngoài

- Sắp xếp trong là sự sắp xếp dữ liệu được tổ chức trong bộ nhớ trong của máy tính, ở đó ta có thể sử dụng khả năng truy nhập ngẫu nhiên của bộ nhớ và do vậy sự thực hiện rất nhanh.
- Sắp xếp ngoài : Sử dụng khi lượng dữ liệu cần sắp xếp lớn không thể lưu trữ trong bộ nhớ trong mà phải lưu trữ trong các tập tin trên bộ nhớ ngoài⇒ Chỉ có thể truy nhập tuần tự, đọc từng phần tử một vào bộ nhớ trong.

2.1.3. Tổ chức dữ liệu và ngôn ngữ cài đặt

- Các đối tượng cần được sắp xếp là các bản ghi gồm một hay nhiều trường, một trong các được gọi là trường khóa(Key), kiểu cuả nó là một kiểu có thứ tự nào đó. Ví dụ:số nguyên, số thực…
- Danh sách các đối tượng cần sắp xếp sẽ là một mảng của các bản ghi nói trên. Mục đích của việc sắp xếp là tổ chức lại các bản ghi sao cho các khóa của chúng được sắp thứ tự tương ứng với quy luật sắp xếp.
- Để trình bày ta sử dụng khai báo sau:

```
const  N=100;
type
        Keytype=Integer;
        Othertype=real;
        Recordtype=Record
                        Key:Keytype;
                        OtherField: Othertype;
                   End;
Var  a:array[1..N] of Recordtype;
Procedure  SWAP(var  x,y:Recordtype);
```

```
Var  Temp: Recordtype;
Begin
        Temp:=x; x:=y; y:=Temp;
End;
```

- Ta thấy thủ tục SWAP lấy O(1) thời gian vì chỉ thực hiện 3 lệnh gán nối tiếp nhau.
- Dãy đích phải thỏa mãn $a_1.key \le a_2.key \le \ldots \le a_n.key$ hoặc ngược lại.

2.1.4. Thuật toán sắp xếp

- Thuật toán sắp xếp gọi là ổn định nếu nó không đảo lộn trật tự ban đầu của các khóa cùng giá trị: nếu $a_i=a_j$, i<j.
- Các thuật toán sắp xếp trong cần đảm bảo:
  + Chỉ sử dụng bộ nhớ trong
  + Có hiệu quả, tiết kiệm bộ nhớ và thời gian.
- Hai phép toán cơ sở khi thực hiện sắp xếp là so sánh và đổi chỗ.
- Thời gian thực hiện thuật giải sẽ đo bằng tổng số lần thực hiện phép so sánh cộng với số lần thực hiện phép đổi chỗ.

2.2. Các phương pháp sắp xếp đơn giản

2.2.1. Sắp xếp chọn (Selection Sort)

Đây là phương pháp đơn giản nhất được tiến hành như sau:

a)Giải thuật:

- Đầu tiên chọn phần tử có khóa nhỏ nhất trong n phần tử a[1] đến a[n] và hoán vị nó với phần tử a[1].
- Chọn phần tử có khóa nhỏ nhất trong n-1 phần tử từ a[2] đến a[n] rồi hoán vị nó với a[2]…
- Ở bước i, chọn phần tử có khóa nhỏ nhất trong n-i+1 phần tử từ a[i] đến a[n] rồi hoán vị nó với a[i].
- Sau n-1 bước thì mảng đã được sắp.

*Ví dụ: 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 2 | 5 6 10 12 9 10 9 3

Bước 3: 2 2 3 | 6 10 12 9 10 9 5

Bước 4: 2 2 3 5 | 10 12 9 10 9 6

Bước 5: 2 2 3 5 6 | 12 9 10 9 10

Bước 6: 2 2 3 5 6 9 | 12 10 9 10

Bước 7: 2 2 3 5 6 9 9 | 10 12 10

Bước 8: 2 2 3 5 6 9 9 10 | 12 10

Kết quả: Bước 9: 2 2 3 5 6 9 9 10 10 | 12

b)Chương trình:

```
Procedure  SelectionSort;
 Var  i,j,k:Integer;  min:Real;
 Begin
     1.for i:=1 to n-1 do
     Begin
       2.  k:=i ; min:=a[i];
       3.   for j:=i+1 to n do
       4.  If  a[j]<min  then  {duyệt các phần tử  2 → n }
             Begin
              5.  min:=a[j];      {phần tử nhỏ nhất từ j → n }
              6.  k:=j ;         {vị trí phần tử nhỏ nhất}
             End;
       7.  Swap(a[i],a[k]);    {Đổi chỗ phần tử nhỏ nhất tìm được với a[i]}
     End;
  End;
```

c)Đánh giá : Các lệnh gán lấy O(1) thời gian, Swap ~ O(1), vòng lặp for 4 thực hiện n-i lần (j chạy i+1 → n) mỗi lần lấy O(1)⇒Lấy O(n-i) thời gian

⇒ Thời gian tính toán là $T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}\approx O(n^2)$

2.2.2. Sắp xếp chèn (InsertionSort)

a)Giải thuật:

Ý tưởng : Lấy dần từng phần tử từ dãy nguồn, chèn vào dãy đích sao cho đảm bảo dãy đích có thứ tự . Xem phần tử a[1] là một dãy đã có thứ tự.

- Bước 1: Chèn phần tử a[2] vào danh sách đã có thứ tự a[1] sao cho a[1], a[2] là một danh sách có thứ tự .

- Bước 2: Chèn phần tử a[3] vào danh sách đã có thứ tự a[1], a[2] sao cho a[1], a[2], a[3] là một danh sách có thứ tự ,…

- Bước i: Chèn phần tử a[i+1] vào danh sách đã có thứ tự a[1], a[2],..,a[i] sao cho a[1],a[2],..,a[i+1] là một danh sách có thứ tự .

Phần tử đang xét a[j] sẽ được chèn vào vị trí thích hợp trong danh sách các phần tử đã được sắp trước đó a[1], a[2],…a[j-1] bằng cách so sánh a[j] với a[j-1] đứng ngay trước nó. Nếu a[j] < a[j-1] thì đổi chỗ a[j] với a[j-1] và tiếp tục so sánh a[j-1] với a[j-2] …

Lặp cho đến khi hết dãy i=n. Mảng được sắp xếp xong.

*Ví dụ:

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)

b)Chương trình:

```
Procedure  InsertionSort;
Var  i,j:Integer;
  Begin
        1.For i:=2 to n do
            Begin
                2.  j:=i;
                3.  While (j>1) and (a[j]<a[j-1]) do
                Begin
                 4.Swap  (a[j],a[j-1]);
                 5.j:=j-1;
                End;
            End;
  End;
```

c)Đánh giá: - Các lệnh (4), (5) đều lấy O(1).

- Vòng lặp (3) chạy nhiều nhất i-1 lần, mỗi lần tốn O(1)$\Rightarrow$(3) lấy i-1 thời gian.

- Lệnh (2), (3) nối tiếp nhau, lệnh (2) lấy O(1)$\Rightarrow$ Cả 2 lệnh lấy i-1

- Vòng lặp (1) có i chạy từ 2$\rightarrow$n$\Rightarrow$ nếu gọi T(n) là thời gian để sắp n phần tử$\Rightarrow$ $T(n)=\sum_{i=2}^{n}(i-1)=\frac{n(n-1)}{2}\approx O(n^2)$

2.2.3. Sắp xếp nổi bọt(Bubble Sort)

a) Giải thuật: Coi các bản ghi được lưu trong một mảng dọc. Qua quá trình sắp, bản ghi nào có khóa "nhẹ" hơn sẽ nổi lên trên. Duyệt toàn mảng, từ dưới lên trên. Nếu 2 phần tử ở cạnh nhau mà không đúng thứ tự tức là phần tử "nhẹ hơn" ở dưới thì phải cho nó "nổi lên" bằng cách đổi chỗ 2 phần tử này cho nhau. Cụ thể:

+ Bước 1: Xét các phần tử từ a[n] đến a[2], với mỗi phần tử a[j] so sánh khóa của nó với khóa của phần tử a[j-1] đứng ngay trước nó. Nếu khóa của a[j] nhỏ hơn khóa của a[j-1] thì đổi chỗ của a[j] và a[j-1].

+ Bước 2: Xét các phần tử từ a[n] đến a[3], làm tương tự

+ Bước i: Xét các phần tử từ a[n] đến a[i+1]…

Sau n bước ta được mảng đã sắp thứ tự.

b)Ví dụ:

Ban đầu	Bước 1	Bước 2	Bước 3	Bước 4
5	2	2	2	2
6	5	2	2	2
2	6	5	3	3
2	2	6	5	5
10	3	3	6	6
12	10	9	9	9
9	12	10	9	9
10	9	12	10	10
9	10	9	12	10
3	9	10	10	12

c)Chương trình:

Procedure BubbleSort;

Var i,j: Integer;

Begin

```
{1}    For i:=1 to n-1 do
{2}      For j:= n downto i+1 do
{3}        if a[j]<a[j-1] then
{4}          Swap(a[j],a[j-1]);
End;
```

d)Đánh giá:

- Lệnh {1} tốn O(1)
- Lặp {2} thực hiện (n-i) lần

} $\Rightarrow$ O(n-i)

$\Rightarrow$ {1} thực hiện (n-1) lần $\Rightarrow T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}=O(n^2)$

2.3. Sắp xếp nhanh QUICK SORT(*phân đoạn Partition Sort*):

{do A.R Hoare phát minh năm 1960}

2.3.1. Ý tưởng

Xét mảng A các bản ghi a[1],..,a[n].

- Chọn một trong các thành phần của mảng làm chốt (Pivot). Phân hoạch mảng thành 2 phần bằng cách chuyển tất cả các thành phần có khóa > chốt sang phải chốt, các thành phần có khóa $\leq$ chốt sang trái chốt $\Rightarrow$ Kết quả của phân hoạch, chốt đứng ở vị trí k và mọi thành phần của mảng con bên trái chốt A[1..k-1] có khóa $\leq$ chốt, mọi thành phần của mảng con bên phải chốt A[k+1..n] có khóa > chốt.

- Sắp xếp độc lập 2 mảng con A[1,..k-1], A[k+1,..,n] bằng cách gọi đệ quy thuật toán trên.

2.3.2. Thiết kế giải thuật

```
Procedure  Quicksort( i,j:Integer);
Var  k:Integer;
Begin
     If i<j then
        Begin
           Partition(i,j,k);  {phân hoạch 2 mảng con A[i,..,k-1] và A[k+1,..,j]}
           Quicksort(i,k-1);
           Quicksort(k+1,j);
        End;
End;
```

Xây dựng thủ tục Partition:

- Vấn đề chọn chốt: Nếu chọn được phần tử làm chốt sao cho kết quả phân hoach nhận được 2 mảng con cân bằng là tốt nhưng sẽ tiêu tốn nhiều thời gian không cần thiết ⇒ Thường chọn phần tử đầu tiên của mảng làm chốt, tức là lấy p=A[1] làm chốt.
- Vấn đề phân hoạch: Sử dụng 2 biến :
  + Biến L chạy từ trái sang phải bắt đầu từ i.
  + Biến R chạy từ phải sang trái bắt đầu từ j+1.

Biến L được tăng cho tới khi A[L] > p, còn biến R được giảm cho tới khi A[R] ≤ p.

Nếu L<R thì đổi chỗ A[L] và A[R]

Lặp lại quá trình trên cho đến khi L>R.

Cuối cùng đổi chỗ A[i] và A[R] để đặt chốt vào đúng vị trí.

```
Procedure  PARTITION ( i,j:Integer; var R:Real);
Var  L:Integer;
      P: kiểu phần tử mảng;
Begin
        P:=A[i]; L:=i; R:=j+1;
        Repeat L:=L+1 until (A[L] > p) or (L > j);
        Repeat R:=R-1 until A[R] ≤ p;
        While  L <R do
          Begin
              Swap(A[L],A[R]);
              Repeat L:=L+1 until A[L] > p;
              Repeat R:=R-1 until A[R] ≤ p;
          End;
        SWAP(A[i],A[R]);
End;
```

Ví dụ : Phân hoạch mảng các số nguyên A[1..8] như sau:

1	2	3	4	5	6	7	8
10	15	4	11	6	3	5	14

Lấy chốt p=A[1]=10, L=1, R=9

Tăng L:=L+1 , giảm R:=R-1 cho tới khi A[L] >10 và A[R]≤ 10

→ Ta có L=2 và R=7. Vì L <R ⇒ Đổi chỗ A[2] cho A[7] ta được.

1 2 3 4 5 6 7 8

10	5	4	11	6	3	15	14
	↑ L					↑ R	

Tiếp tục tăng L và giảm R cho tới khi A[L] >10 và A[R]≤ 10.

⇒Ta có L=4 và R=6. Vì L <R ⇒Đổi chỗ A[4] , A[6] ,ta được :

1	2	3	4	5	6	7	8
10	5	4	3	6	11	15	14
			↑ L		↑ R		

Tiếp tục tăng L , giảm R cho tới khi A[L] >10 và A[R]≤10

⇒Ta có L=6 và R=5 . Vì L>R, đổi chỗ A[1] và A[5]

Đặt chốt p=A[1] vào vị trí R=5 , ta được:

1	2	3	4	5	6	7	8
6	5	4	3	10	11	15	14
				↑ R	↑ L		

Như vậy mảng A[1..8] được phân hoạch thành 2 mảng con A[1..R-1] và A[R+1..8] tức A[1..4] và A[6..8]. Sau đó tiếp tục QUICKSORT 2 mảng con trên ta sẽ được mảng đã sắp xếp.

2.3.3. Đánh giá độ phức tạp

- Thời gian thực hiện thủ tục Partition là thời gian đi qua mảng(hai biến L,R chạy từ 2 đầu cho tới khi chúng gặp nhau) và so sánh khóa của mỗi thành phần mảng với khóa của chốt. Do đó để phân hoạch mảng n phần tử ta cần thời gian O(n)=P(n)

- Gọi T(n) là thời gian thực hiện Quicksort thì T(n) phải là tổng của P(n) và thời gian đệ qui Quicksort cho 2 mảng, còn trong trường hợp xấu nhất là chọn phải phần tử có khóa lớn nhất làm chốt ⇒Việc phân hoạch bị lệch, tức là mảng bên phải chỉ gồm một phần tử chốt, còn mảng bên trái gồm n-1 phần tử còn lại. Khi đó ta có thể thành lập phương trình đệ qui :

$$T(n)=\begin{cases}1 \text{ nếu } n=1\\ O(n)+T(n-1)+T(1) \text{ nếu } n>1\end{cases} \Leftrightarrow T(n)=\begin{cases}1 \text{ nếu } n=1\\ T(n-1)+T(1)+n \text{ nếu } n>1\end{cases}$$

Giải phương trình đệ qui này bằng phương pháp truy hồi:

Ta có: $T(n) = T(n-1) + T(1) + n = T(n-1) + (n+1)$

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

....

$$= T(n-i) + (n-i+2) + (n-i+3) + ... + n + (n+1)$$
$$= T(n-i) + \sum_{j=n-i+2}^{n+1} j$$

Quá trình trên kết thúc khi i=n-1, khi đó ta có:

$$T(n) = T(1) + \sum_{j=3}^{n+1} j = 1 + \frac{(n-1)(n+4)}{2} = \frac{n^2 + 3n - 2}{2} = O(n^2)$$

Trường hợp tốt nhất khi chọn chốt sao cho 2 mảng con có kích thước cân bằng.

$\Rightarrow$ Phương trình đệ quy:

$$T(n) = \begin{cases} 1 \text{ nếu } n=1 \\ 2T(n/2) + n \text{ nếu } n>1 \end{cases}$$

Giải phương trình đệ quy trên ta được T(n) = O(nlogn).

Như vậy trong trường hợp trung bình T(n)=O(nlogn) $\Rightarrow$ Khá hơn các giải thuật trước.

2.4. Sắp xếp kiểu vun đống (Heapsort)

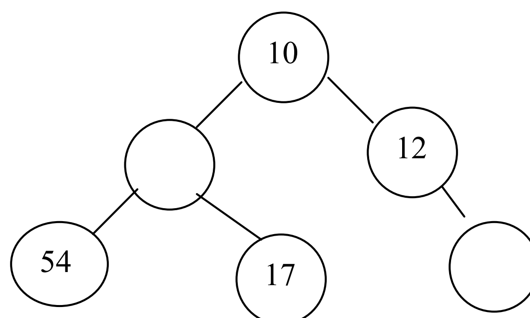
2.4.1. Định nghĩa HEAP

HEAP là một cây nhi phân đầy đủ trái mà mỗi nút được gán một giá trị khóa sao cho giá trị khóa ở nút cha bao giờ cũng nhỏ hơn hoặc bằng giá trị khóa ở 2 nút con. Do đó trong HEAP ta có :

- Nút gốc có khóa bé nhất
- Dãy khóa nhận được khi đi theo một đường bất kì là dãy có thứ tự tăng dần

Người ta còn gọi HEAP là cây nhị phân có thứ tự bộ phận, HEAP - tiếng Anh có nghĩa là "đống". Có thể minh họa HEAP bằng hình ảnh "vun đống" một đống đất đá chẳng hạn, những cục nhỏ nhẹ sẽ nằm trên, những cục nặng hơn sẽ ở dưới.

Ví dụ: Một HEAP



Cấu trúc mảng thích hợp để biểu diễn cây nhị phân kiểu HEAP vì HEAP là cây nhị phân đầy đủ trái. Khi đó ta nói mảng thỏa mãn điều kiện HEAP nếu mảng biểu diễn cây nhị phân là HEAP.

Ví dụ : Mảng biểu diễn cây trên

10	15	12	54	17	91

- Như vậy với "đống": nếu j chỉ vào vị trí của nút con thì [j/2] chỉ vào vị trí của nút cha, hay con của nút i là các nút thứ 2i (con trái) và 2i+1(con phải).
- Nút ứng với chỉ số [n/2] trở xuống mới có thể là cha của các nút khác.

Ví dụ : n=10 $\Rightarrow$ các nút 5, 4, 3, 2, 1 mới có thể là cha.

2.4.2. Sắp xếp kiểu vun đống

Được chia thành 2 giai đoạn:

<u>*Giai đoạn 1*</u> : Tạo đống

Từ cây nhị phân đã cho ta biến đổi để nó trở thành một "đống".

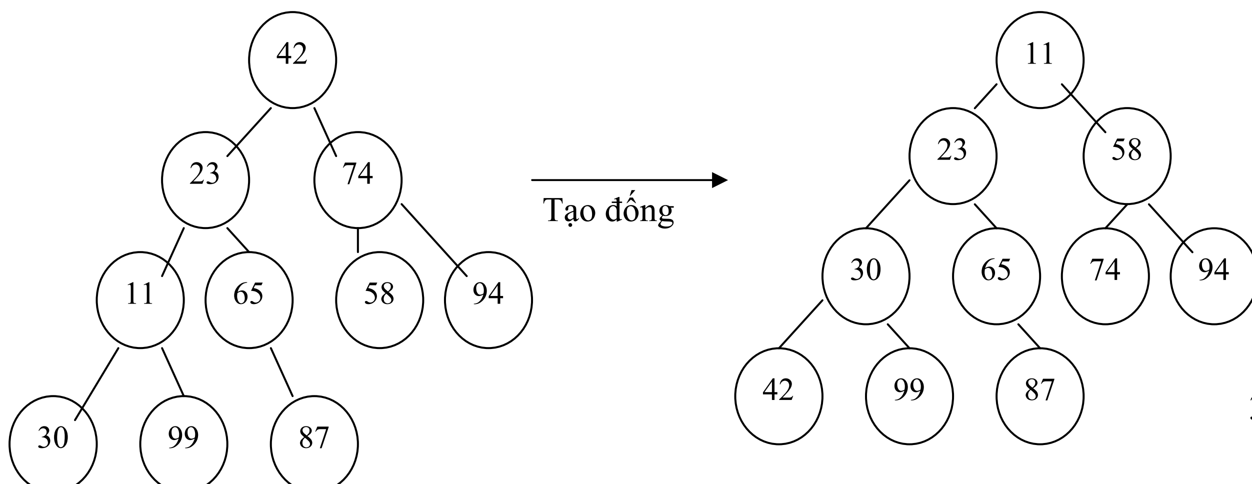
<u>*Giai đoạn 2*</u> : Sắp xếp.

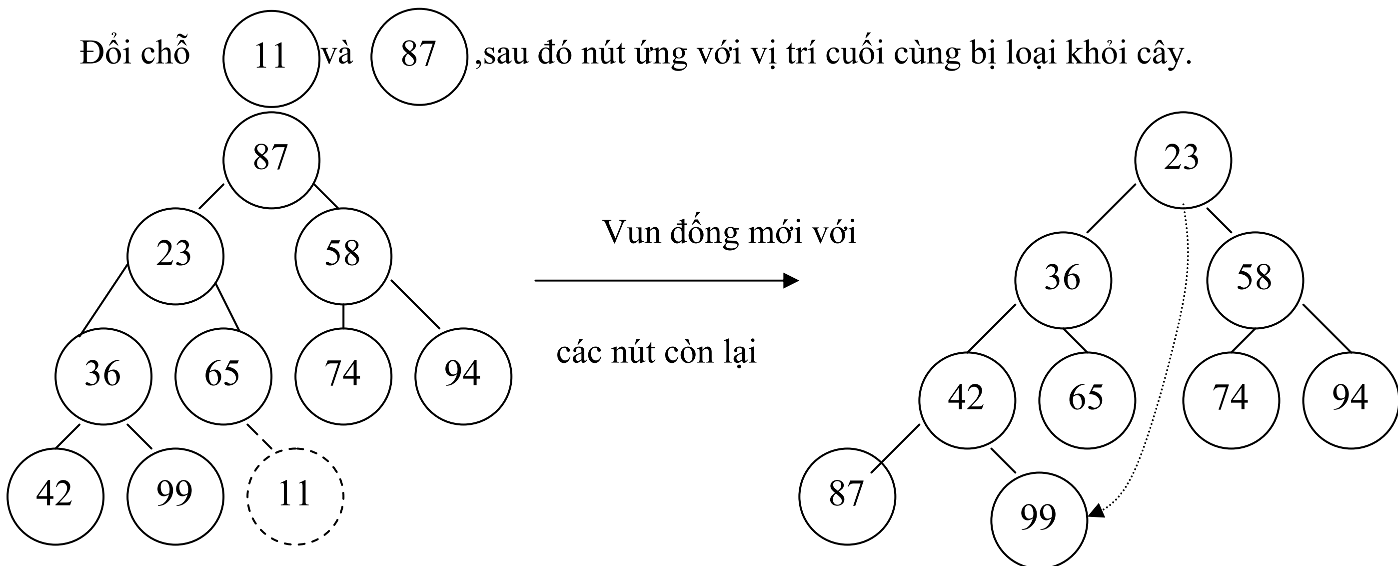
Nhiều lượt xử lý được thực hiện, mỗi lượt ứng với hai phép:

- Đưa khóa nhỏ nhất về vị trí thực của nó bằng cách đổi chỗ A[1] và A[n].
- "Vun lại thành đống" đối với cây gồm các khóa còn lại (sau khi loại khóa nhỏ nhất ra ngoài)

*Ví dụ: Với dãy khóa 42 23 74 11 65 58 94 36 99 87

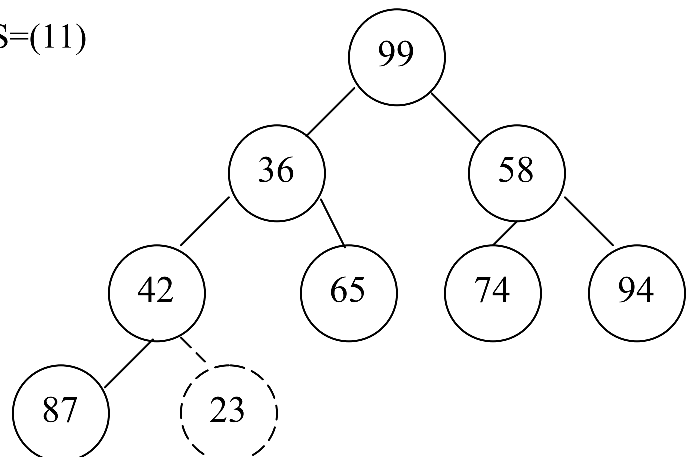
thì cấu trúc ban đầu là cây có dạng:





Dãy được sắp S=(11)

Đổi chỗ 23 với 99



S = (23, 11)

………..

Thiết kế giải thuật:

Vấn đề cần giải quyết trước hết là: Biến đổi mảng ban đầu A[1..n] thành mảng biểu diễn cây thứ tự bộ phận("vun đống") như thế nào. Ta có nhận xét rằng: với i>n div 2 thì điều kiện "đống" xem như thỏa mãn vì 2i và 2i+1 >n. Như vậy nếu ta chỉ xét i chạy từ n div 2 giảm xuống 1, đẩy A[i] xuống vị trí thích hợp trong mảng A[1..n] thì cuối cùng sẽ nhận được A[1..n] thỏa mãn điều kiện "vun đống".

$\Rightarrow$Xây dựng thủ tục Pushdown(a,b) thực hiện việc đẩy A[a] xuống vị trí thích hợp trong mảng A[a..b] để thỏa mãn điều kiện "đống" với $\forall i \geq a$.

+*Thủ tục Pushdown*: Các phần tử thuộc nửa sau của mảng không có con$\Rightarrow$không phạm điều kiện HEAP$\Rightarrow$Chỉ cần xét từ giữa mảng trở về trước.

```
Procedure  PUSHDOWN(a,b:Integer);
Var  i.j: integer;                          {lưu vị trí của nút cha, con}
      OK: Boolean;               {đánh dấu những nút thỏa mãn điều kiện HEAP }
 Begin
      i:=a;  OK:=False;
      1.  While (i≤ b div 2) and not OK do
         { chỉ xét các nút có con và chưa thỏa mãn điều kiện đống}
            Begin
     2.          If i=b div 2 then j:=2*i            { chỉ có con trái}
                  else if A[2*i]<A[2*i+1] then j:=2*i
                       else j:= 2*i+1              {nếu có con trái<con phải thì so sánh con
            trái với cha,ngược lại thì so sánh con phải với cha}
     3.          If A[i]>A[j] then
                    Begin                    { nếu con <cha thì đổi chỗ con với cha}
                       Swap(A[i],A[j]) ;
                        i:=j;                  {đẩy xuống xét tiếp}
                    End;
                 Else OK:=True;
            End;
End;
```

*Sử dụng Pushdown trong thủ tục HEAPSORT. Kết quả được mảng A[1…n] sắp xếp giảm dần.

```
Procedure  HEAPSORT;
Var i: integer;
Begin
    1) for i:=n div 2 downto 1 do PUSHDOWN(i,n);   { biến đổi mảng A[1..n] thành
    mảng HEAP}
    2) for i:=n  downto 2 do
      Begin
        Swap (A[1],A[i]);                  { đổi chỗ gốc xuống dưới cùng}
        PUSHDOWN(1,i-1);                  {vun lại thành đống các phần tử còn lại}
      End;
End;
```

2.4.3. Độ phức tạp tính toán

- Thời gian thực hiện Pushdown

+ Thân của vòng lặp (1) là các lệnh (2) và (3), mỗi lệnh cần O(1)$\Rightarrow$Thân (1) cần O(1).

+ Sau mỗi lần lặp, biến i tăng ít nhất 2 lần, giá trị ban đầu của i là a$\Rightarrow$gọi k là số lần lặp $\Rightarrow a.2^k \le b \Rightarrow k \le \log\left(\frac{b}{a}\right) \Rightarrow$ thời gian thực hiện Pushdown(a,b) là O(log(b/a))

- Thời gian thực hiện Heapsort

+ Vòng (1) lặp $\frac{n}{2}$ lần, mỗi lần gọi Pushdown cần nhiều nhất là O(logn)$\Rightarrow$thời gian thực hiện (1) là O(nlogn);

+ Vòng (2) lặp n-1 lần, mỗi lần lặp cần thực hiện Swap và gọi Pushdown$\Rightarrow$Cần nhiều nhất O(logn)$\Rightarrow$Thời gian thực hiện (2) nhiều nhất O(nlogn)$\Rightarrow$Thời gian thực hiện Heapsort là tổng hai vòng lặp For$\Rightarrow$O(nlogn).

2.5. Sắp xếp hòa nhập (*Merge Sort*)

2.5.1. Ý tưởng: Xuất phát từ chỗ các phương pháp sắp xếp đã xét thường được làm với các file dữ liệu không lớn. Với các file lớn người ta thường phân ra thành các file nhỏ hơn để sử dụng các thuật toán trên. Như vậy kết quả sẽ có nhiều phần tử được sắp xếp theo cùng một thứ tự. Thuật toán Merge sẽ hòa nhập các file này thành một file có độ dài như file ban đầu, bằng cách tiến hành từng 2 file một, kết quả hòa tiếp với file thứ 3 và tiếp tục như vậy đến hết.

Giả sử có 2 dãy X={x_1,x_2,..,x_t} đã sắp xếp tăng

dãy Y={y_1,y_2,..,y_s} đã sắp xếp tăng

Thực hiện hòa nhập X và Y sao cho vẫn thỏa mãn điều kiện (tăng dần)

- So sánh giữa 2 khóa nhỏ nhất của 2 dãy X và Y, chọn khóa nhỏ hơn đưa vào dãy mới Z, đồng thời loại bỏ nó khỏi dãy ban đầu.

- Quá trình so sánh lặp lại cho đến khi 1 trong 2 dãy X và Y cạn. Phần còn lại của dãy kia xếp nốt vào phần sau của dãy Z.

*Ví dụ : X={15, 20, 50, 76}

Y={8, 17, 92}

+Bước 1: Z={8} , X={15, 20, 50, 76}

Y={ 17, 92}

+Bước 2: Z={8,15}, X={ 20, 50, 76}

Y={ 17, 92}

+Bước 3: Z={8,15,17} , X={20, 50, 76}

Y={92}

+Bước 4: Z={8,15,17,20}, X={50, 76}

Y={92}

+Bước 5: Z={8,15,17,20,50} , X={76}

Y={92}

+Bước 6: Z={8,15,17,20,50,76}….

2.5.2. Thiết kế giải thuật

a) Thủ tục MERGE để hòa nhập 2 dãy khóa đã sắp xếp $X=\{x_1,x_2,\ldots,x_m\}$, $Y=\{y_1,y_2,\ldots,y_n\}$, Z là dãy kết quả có độ dài không lớn hơn n+m.

```
Procedure   MERGE (X,Y,Z,m,n);
Var  i,j,k : integer;
Begin
        i:=1 ; j:=1 ; k:=1 ;
        While (i<m) and (j<n) do
        begin
         If X[i]<Y[j] then
           Begin
              Z[k]:= X[i];
              i:=i+1;
           End
         else
           Begin
              Z[k]:=Y[j];
              j:=j+1;
           End;
         k:=k+1;
        end;
        If i>m then
          Begin              { hết dãy X,nối phần còn lại của dãy Y vào Z}
             Z[k]:=Y[j];
             For q:=1 to n-j do Z[k+q]:=Y[j+q];
```

```
        End;
      Else
        Begin            { hết dãy Y,nối phần còn lại của dãy X vào Z}
          Z[k]:=X[i];
          For q:=1 to m-i do  Z[k+q]:=X[j+q];
        End;
End;
```

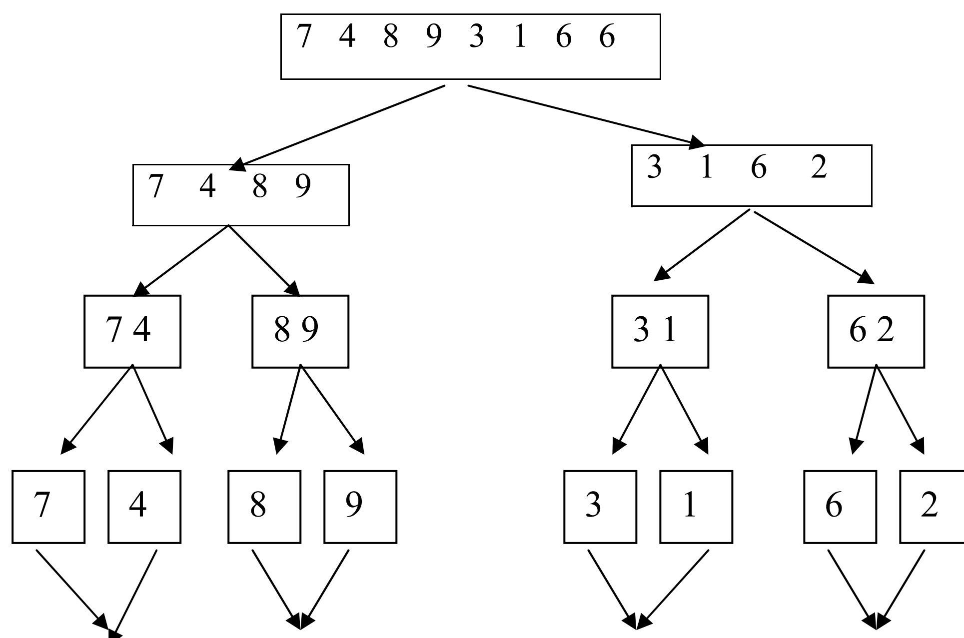
b) Thủ tục MERGESORT:

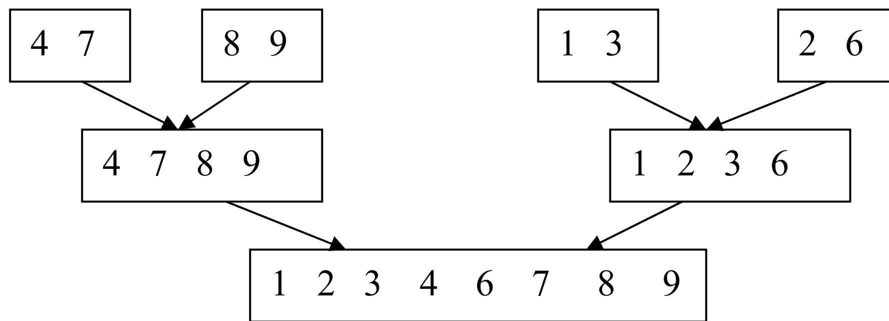
A là dãy khóa, B là dãy chứa kết quả đã sắp xếp, p, q tương ứng là hai chỉ số đầu và cuối của dãy A.

```
Procedure  MERGESORT (A,B,p,q);
Var  t : integer;
Begin
  If p<q then
    Begin
        t:= [(p+q)/2];
        MERGESORT (A,B1,p,t);
        MERGESORT (A,B2,t+1,q);
        MERGE (B1,B2,B,t-p+1,q-t);
    End;
End;
```

* Ví dụ: Sắp xếp danh sách A gồm 8 phần tử 7, 4, 8, 9, 3, 1, 6, 2.

Mergesort như sau:





2.5.3. **Đánh giá độ phức tạp:** O(nlogn) *(Đã đánh giá ở phần trước*).

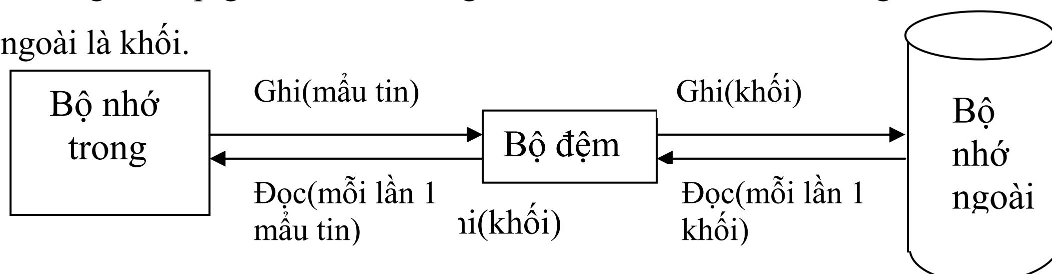
2.6. Cấu trúc dữ liệu và giải thuật xử lý ngoài

2.6.1. Mô hình xử lý ngoài

Khi số lượng dữ liệu vượt quá khả năng lưu trữ của bộ nhớ trong (xử lý phiếu điều tra dân số toàn quốc, quản lý đất đai..), ta phải dùng bộ nhớ ngoài để lưu trữ và xử lý. Các thiết bị lưu trữ ngoài như băng từ, đĩa từ đều có khả năng lưu trữ lớn nhưng đặc điểm truy nhập hoàn toàn khác bộ nhớ trong. Do đó, phải tìm CTDL> thích hợp cho việc xử lý dữ liệu lưu trữ trên bộ nhớ ngoài.

- Kiểu dữ liệu tập tin là kiểu thích hợp nhất cho việc biểu diễn dữ liệu được lưu trên bộ nhớ ngoài. Hệ điều hành chia bộ nhớ ngoài thành các khối (block) có kích thước bằng nhau (từ 512 bytes đến 4096 bytes).
- Trong quá trình xử lý việc chuyển giao dữ liệu giữa bộ nhớ trong và bộ nhớ ngoài được tiến hành thông qua vùng nhớ đệm (buffer). Bộ đệm là một vùng dành riêng cho bộ nhớ trong có kích thước bằng kích thước một khối của bộ nhớ ngoài.
- Có thể xem 1 tập tin bao gồm nhiều mẩu tin được lưu trong các khối. Mỗi khối lưu một số nguyên vẹn các mẩu tin.
- Trong thao tác đọc, nguyên một khối của tập tin được chuyển vào trong bộ đệm, lần lượt đọc các mẩu tin có trong bộ đệm cho tới khi bộ đệm rỗng thì lại chuyển một khối từ bộ nhớ ngoài vào bộ đệm.
- Để ghi thông tin ra bộ nhớ ngoài, các mẩu tin được xếp vào bộ đệm cho đến khi đầy bộ đệm thì một khối được chuyển ra bộ nhớ ngoài, khi bộ đệm rỗng thì xếp tiếp các mẩu tin vào.

⇒Đơn vị giao tiếp giữa bộ nhớ trong và bộ đệm là mẩu tin, còn giữa bộ đệm và bộ nhớ ngoài là khối.



Hình: Mô hình giao tiếp giữa bộ nhớ trong, bộ nhớ ngoài và vùng nhớ đệm

2.6.2. Đánh giá các giải thuật xử lý ngoài

- Đối với bộ nhớ ngoài, thời gian tìm một khối để đọc vào bộ nhớ trong là rất lớn so với thời gian thao tác trên dữ liệu trong khối đó.

Ví dụ: Giả sử một khối lưu 1000 số nguyên được lưu trên đĩa quay với vận tốc 1000 v/phút thì thời gian để đưa đầu từ vào rãnh chứa khối và quay đĩa để đưa khối đến chỗ đầu từ hết khoảng 100 miligiây. Với thời gian này máy có thể thực hiện 100000 lệnh, đủ để sắp xếp các số nguyên này theo giải thuật Quicksort⇒ Khi đánh giá các giải thuật thao tác trên bộ nhớ ngoài, ta tập trung vào việc xét số lần đọc khối vào bộ nhớ trong và số lần ghi khối ra bộ nhớ ngoài gọi là phép truy xuất khối (*block access*). Vì kích thước các khối là cố định nên ta không thể tìm cách tăng kích thước một khối mà phải tìm cách giảm số lần *truy xuất khối.*

2.6.3. Sắp xếp ngoài - MergeSorting

Sắp xếp dữ liệu được tổ chức như một tập tin hoặc tổng quát hơn, sắp xếp dữ liệu được lưu trên bộ nhớ ngoài gọi là sắp xếp ngoài.

a)Khái niệm đường

- Đường độ dài k là một tập hợp k mẩu tin được sắp thứ tự theo khóa tức là nếu các mẩu tin r_1, r_2,.., r_k lần lượt có khóa là k_1, k_2,.., k_k tạo thành một đường thì $k_1 \le k_2 \le \ldots \le k_k$.

- Cho tập tin chứa các mẩu tin r_1, r_2,.., r_n, ta nói tập tin được tổ chức thành đường có độ dài k nếu ta chia tập tin thành các đoạn k mẩu tin liên tiếp và mỗi đoạn là một đường, đoạn cuối có thể không đủ k mẩu tin, ta gọi đoạn này là đuôi (tail).

*Ví dụ : Tập tin gồm 14 mẩu tin có khóa là các số nguyên được tổ chức thành 4 đường độ dài 3 và 1 đuôi độ dài 2 :

5	6	9	13	26	17	1	5	8	12	14	27	23	25

b)Giải thuật

Để sắp tập tin F có n mẩu tin ta sử dụng 4 tập tin F_1, F_2, G_1, G_2.

- Phân phối các mẩu tin của tập tin đã cho F luân phiên vào 2 tập tin F_1, F_2. Như vậy 2 tập tin này được xem như tổ chức thành đường độ dài 1.

+ Bước 1 : Đọc 2 đường, mỗi đường có độ dài 1 từ 2 tập tin F_1,F_2 và trộn 2 đường nay thành một đường có độ dài 2 và ghi luân phiên vào 2 tập tin G_1, G_2. Đổi vai trò F_1 cho G_1, F_2 cho G_2.

+ Bước 2 : Đọc 2 đường, mỗi đường có độ dài 2 từ 2 tập tin F_1, F_2 và trộn hai đường này thành một đường có độ dài 4 và ghi luân phiên vào 2 tập tin G_1, G_2 . Đổi vai trò của F_1 cho G_1, F_2 cho G_2.

Quá trình trên tiếp tục và sau i bước thì độ dài một đường là 2^i. Nếu $2^i \geq n$ thì giải thuật kết thúc, lúc đó tập tin G_2 rỗng và G_1 chứa các mẩu tin đã được sắp.

c)Độ phức tạp tính toán

Giải thuật kết thúc sau i bước với $i \geq \log n$. Mỗi bước phải đọc từ 2 tập tin và ghi vào 2 tập tin, mỗi tập tin có trung bình n/2 mẩu tin. Giả sử mỗi khối lưu được b mẩu tin thì mỗi bước cần đọc và ghi (2*2*n)/(2*b) = 2n/b khối, mà cần logn bước $\Rightarrow$Cần $\frac{2n}{b}\log n$ phép truy nhập khối (*block access).*

Ví dụ : Cho tập tin F có 23 mẩu tin với khóa là các số nguyên

F : 28, 31, 3, 5, 93, 27, 15, 10, 40, 65, 9, 30, 39, 13, 90, 8, 10, 77, 8, 20, 69, 23

Phân phối các mẩu tin của F luân phiên vào 2 tập tin F_1, F_2 tổ chức thành các đường dộ dài 1 :

28	3	93	15	40	9	39	90	10	8	22	23	F_1

31	5	27	10	65	30	13	8	77	20	69	F_2

+Bước 1 : Trộn các đường có độ dài 1 của F_1, F_2, được các đường có độ dài 2, ghi luân phiên vào 2 tập G_1, G_2

G_1	28 31	27 93	40 65	13 39	10 77	22 69	F_1

G_2	3 5	10 15	9 30	8 90	8 20	23	F_2

+Bước 2 : Đổi vai trò G_1cho F_1, G_2 cho F_2. Trộn các đường độ dài 2 trong F_1, F_2 được các đường độ dài 4 ghi luân phiên vào G_1, G_2.

G_1	3 5 28 31	9 30 40 65	8 10 20 77	F_1

G_2 | 10 15 27 93 | 8 13 39 40 | 22 23 69 | F_2

+Bước 3 :

G_1 | 3 5 10 15 27 28 31 93 | 8 10 20 22 23 69 77 | F_1

G_2 | 8 9 13 30 39 40 65 90 | F_2

+Bước 4 :

G_1 | 3 5 8 9 10 13 15 27 28 30 31 39 40 65 90 93 | F_1

G_2 | 8 10 20 22 23 69 77 | F_2

+Bước 5 :

| 3 5 8 8 9 10 13 15 20 22 23 27 28 30 31 39 40 65… |

Procedure MERGE (K : integer; f_1,f_2,g_1,g_2 : file of Recordtype);

{Trộn cac đường có độ dài k trong F_1,F_2 thành các đường có độ dài 2k và ghi luân phiên vào 2 tập G_1,G_2}

Var

Ghitập: Boolean; *{nếu Ghitập=True thì ghi vào G_1, ngược lại ghi vào G_2}*

Ghimẩu : Integer; *{mẩu tin hiện hành nào trong 2 tập F_1,F_2 sẽ được ghi ra G_1 hoặc G_2}*

Used: array[1..2] of Integer;*{ghi số mẩu tin đã được đọc trong đường hiện tại của F_j}*

Fin: array[1..2] of Boolean; *{Fin(j)=True nếu đã đọc hết các mẩu tin trong đường hiện hành của F_j hoặc đã đến cuối F_j }*

Current: array[1..2] of Recordtype; *{Current(j) lưu mẩu tin hiện hành của tập F[j]}*

Procedure GetRecord(i:Integer);

{ Nếu đã đọc hết các mẩu tin trong đường hiện hành của f_i hoặc cuối f_i thì đặt Fin[i]=True nếu không thì đọc một mẩu tin của tập f_i vào Current[i]}

```
Begin
    Used[i]:=Used[i]+1;
    If(Used[i]=k+1) or (i =1) and (eof(f1)) or (i=2) and (eof(f2)) then Fin[i]:=True
    Else if i =1 then Read(f1, current[2]);
    Else Read(f2, current[2]);
End;
Begin  {Khởi tạo}
      Ghitập:=True;    {ghi vào tập F1}
      Reset(f1); Reset(f2);
      Rewrite(g1); Rewrite(g1);
      While (not eof(f1)) or (not eof(f2)) do
       { Bắt đầu đọc các mẩu tin từ trong 2 đường hiện hành của f1,f2}
      Begin
          Used[1]:=0 ; Used[2]:=0;
          Fin[1]:=False ; Fin[2]:=False;
          Getrecord(1); Getrecord(2);
          While ( not fin[1]) or (not fin[2]) do
          Begin      { trộn 2 đường , chọn mẩu tin sẽ được ghi }
              If Fin[1] then Ghimẩu:=2
              Else if Fin[2] then Ghimẩu:=1
                   Else if current[1].Key< current[2].Key then Ghimẩu:=1
                        Else Ghimẩu:=2;
              If Ghitập then Write(g1,current(Ghimẩu))
              Else Write(g2,current(Ghimẩu));
              Getrecord(Ghimẩu);
          End;
          Ghitập:=Not Ghitập;
      End;
End;
```

2.6.4. Cải tiến sắp xếp trộn

- Quá trình sắp xếp trộn nói trên bắt đầu từ các đường độ dài 1 nên sau logn bước giải thuật mới kết thúc.

- Ta có thể tiết kiệm thời gian bằng cách chọn một số k thích hợp sao cho k mẩu tin có thể đủ chứa trong bộ nhớ trong. Mỗi lần đọc vào bộ nhớ trong k mẩu tin, dùng sắp xếp trong (Quicksort,...) để sắp xếp k mẩu tin này và ghi luân phiên vào 2 tập F_1,F_2 sau đó tiến hành sắp xếp trộn với các tập tin được tổ chức thành các đường độ dài k.

- Sau i bước thì độ dài mỗi đường là $k.2^i$. Giải thuật kết thúc khi $k.2^i \geq n$ hay $i \geq \log(n/k)$ $\Rightarrow$ Số phép truy xuất khối là $\frac{2n\log(n/k)}{b} < \frac{2n\log n}{b}$ $\Rightarrow$ Tăng tốc độ sắp xếp trộn.

*Ví dụ : Lấy tập tin F như ví dụ 1

Giả sử bộ nhớ trong có thể chứa được 3 mẩu tin, đọc lần lượt 3 mẩu tin của F vào bộ nhớ trong, dùng một sắp xếp trong để sắp xếp và ghi luân phiên vào F_1 và F_2.

3 28 31	10 15 40	13 39 90	8 20 22	F_1

5 27 93	9 30 65	8 10 77	23 69	F_2

+Bước 1 : Trộn các đường độ dài 3 của F_1 và F_2 được các đường độ dài 6, ghi luân phiên vào G_1, G_2.

G_1	3 5 27 28 31 93	8 10 13 39 77 90	F_1

G_1	9 10 15 30 40 65	8 20 22 23 69	F_2

+Bước 2 : Đổi vai trò F_1 cho G_1, F_2 cho G_2, trộn 2 đường…

G_1	3 5 9 10 15 27 28 30 31 40 65 93	F_1

G_2	8 8 10 13 20 22 23 39 69 77 90	F_2

+Bước 3 :

G_1	3 5 8 8 9 10 10 13…

2.6.5. Trộn nhiều đường(*Multiway Merge*)

a) Giải thuật

Để sắp xếp tập F có n mẩu tin ta sử dụng m tập tin (m chẵn): F[1], F[2],…, F[m] *(nếu m = 4 ta có giải thuật sắp xếp trộn bình thường).*

Đặt h = m/2, giả sử bộ nhớ trong có thể chứa k mẩu tin.

- Khởi đầu: Mỗi lần đọc từ tập tin F vào bộ nhớ trong k mẩu tin sử dụng một sắp xếp trong để sắp xếp k mẩu tin này thành đường rồi ghi luân phiên vào các tập tin F[1], F[2], …, F[h].
- Bước 1: Trộn các đường độ dài k của h tập tin F[1], F[2],…, F[h] thành đường có độ dài k.h ghi luân phiên vào h tập tin F[h+1], F[h+2],…F[m]. Đổi vai trò của F[i] và F[h+i] ($1 \le i \le h$). Sau i bước thì độ dài mỗi đường là $k.h^i$ và giải thuật kết thúc khi $k.h^i \ge n \Rightarrow$ Tập tin đã được sắp là một đường ghi trong F[h+1].

b) Độ phức tạp tính toán

Giải thuật kết thúc sau i bước, $k.h^i \ge n$ hay $i \ge \log_h(n/k)$. Mỗi bước ta phải đọc từ h tập tin và ghi vào h tập tin, trung bình mỗi tập tin có n/h mẩu tin.

Giả sử mỗi khối lưu được b mẩu tin thì mỗi bước phải truy xuất $\frac{2.2.n}{2.b} = \frac{2n}{b}$ khối.

Vì cần $\log_h(n/k)$ bước $\Rightarrow$ cần $\frac{2n \log_h \frac{n}{k}}{b}$ phép truy xuất khối, rõ ràng $\log_h(n/k) < \log(n/k)$ và thủ tục Mergesort là một trường hợp đặc biệt khi h = 2.

Ví dụ: Cho tập tin F như ví dụ trên

Sử dụng 6 tập tin để sắp xếp F. Giả sử bộ nhớ trong có thể chứa 3 mẩu tin, đọc lần lượt 3 mẩu tin của F vào bộ nhớ trong, dùng một sắp xếp trong để sắp xếp và ghi luân phiên vào 3 tập tin F[1], F[2], F[3]

F_1	3 28 31	9 30 65	8 20 22
F_2	5 27 93	13 39 90	23 69
F_3	10 15 40	8 10 77	

Bước 1: Trộn các đường độ dài 3 trong F_1, F_2, F_3 thành các đường độ dài 9 ghi vào F_4, F_5, F_6.

F_4	3 5 10 15 27 28 31 40 93	F_1

F_5 | 8 9 10 13 30 39 65 77 90 | F_2

F_6 | 8 20 22 23 69 | F_3

Bước 2: Đổi vai trò F_1 cho F_4, F_2 cho F_5, F_3 cho F_6 . Trộn các đường độ dài 9 trong F_1, F_2, F_3 thành đường độ dài 23 ghi vào F_4.

F_4 | 3 5 8 8 9 10 10… |

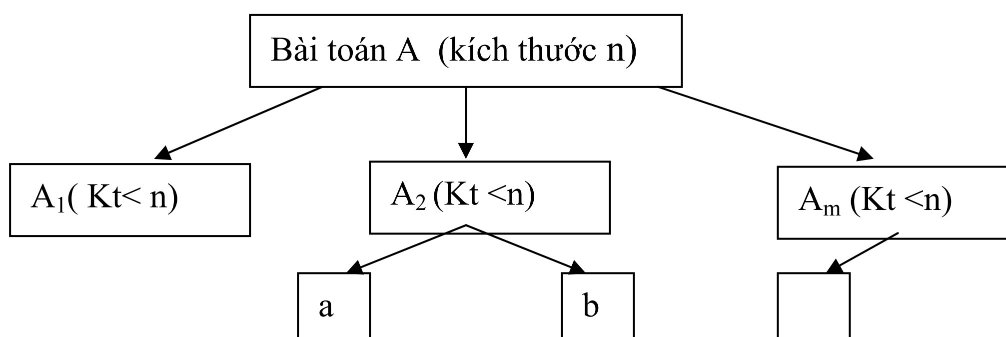
Chương 3
KỸ THUẬT THIẾT KẾ THUẬT TOÁN

Mặc dù không tồn tại một phương pháp vạn năng có thể giúp ta thiết kế được thuật toán giải quyết mọi vấn đề, nhưng các nhà nghiên cứu đã tìm ra một số phương pháp thiết kế thuật toán cơ bản còn gọi là các chiến lược thiết kế thuật toán. Đó là các phương pháp: Chia để trị (*Divide – and - Conquer*), quy hoạch động (*dynamic programming*), kỹ thuật tham ăn (*greedy method*), quay lui (*backtracking*), nhánh và cận (*branch – and – bound*), tìm kiếm địa phương (*local search*)…

Các kỹ thuật này được áp dụng vào một lớp rộng các bài toán trong đó có những bài toán nổi tiếng như: Bài toán tìm đường đi ngắn nhất của người đi giao hàng, bài toán về các chu trình… Tuy nhiên cần lưu ý rằng, các phương pháp trên chỉ là các chiến lược có tính định hướng sự tìm tòi thuật toán. Việc áp dụng một số chiến lược nào đó để tìm ra thuật toán cho một bài toán còn đòi hỏi nhiều sáng tạo.

3.1. Chia để trị

3.1.1. Nội dung: Đây là kỹ thuật từ trên xuống (top – down), là kỹ thuật quan trọng nhất, được áp dụng rộng rãi nhất để thiết kế các giải thuật có hiệu quả.



Trong đó:

- A_1, A_2,…là các bài toán con cùng dạng với bài toán A, kích thước nhỏ hơn kích thước bài toán A.
- a, b,… : Các bài toán cơ sở có lời giải hiển nhiên thuộc loại dễ dàng thực hiện.

Tóm lại kỹ thuật chia để trị gồm 2 quá trình:

+ Phân tích bài toán đã cho thành các bài toán cơ sở.
+ Tổng hợp kết quả từ các bài toán cơ sở để có lời giải cho bài toán ban đầu.

- Đối với một số bài toán, quá trình phân tích đã chứa đựng việc tổng hợp kết quả$\Rightarrow$nếu ta giải xong các bài toán cơ sở thì bài toán ban đầu cũng đã được giải quyết. Ngược lại có những bài toán mà quá trình phân tích thì đơn giản nhưng việc tổng hợp kết quả lại rất phức tạp.
- Kỹ thuật này sẽ cho ta một giải thuật đệ quy mà việc xác định độ phức tạp phải giải một phương trình đệ quy nào đó.

Lược đồ phương pháp chia để trị:

Procedure DivideConquer(A,x); {*Tìm nghiệm x của bài toán A}*

Begin

If A đủ nhỏ **then Solve(A)**

Else begin

Phân A thành các bài toán con $A_1, A_2, \ldots, A_m$;

For i:=1 **to** m **do DivideConquer(A_i,x_i);**

Kết hợp các nghiệm x_i (i = 1, 2, …, m) của các bài toán con A_i để nhận được nghiệm x của bài toán A

End;

End;

3.1.2. Một số bài toán áp dụng

a) Giải thuật Mergesort và Quicksort

- Với Mergesort, kỹ thuật "*Divide and conquer*" thể hiện ở quá trình chia đôi một danh sách, quá trình này sẽ dẫn đến bài toán sắp xếp một danh sách có độ dài bằng 1 *(bài toán cơ sở*). Việc tổng hợp kết quả ở đây là "*trộn*" 2 danh sách đã sắp để được một danh sách có thứ tự.

- Với Quicksort, kỹ thuật "*Divide and conquer*" thể hiện ở quá trình phân hoạch danh sách thành 2 danh sách con "*bên trái*" và "*bên phải*", sắp xếp "*bên trái*" và "*bên phải*" để được danh sách có thứ tự. Quá trình phân chia dẫn đến các bài toán sắp xếp một danh sách chỉ gồm một phần tử hoặc nhiều phần tử có khóa bằng nhau (bài toán cơ sở). Với Quicksort không phải tổng hợp kết quả vì việc đó đã thực hiện trong quá trình phân hoạch.

b) Bài toán nhân các số nguyên lớn:

Cho 2 số nguyên n chữ số X, Y,

- Theo giải thuật nhân 2 chữ số thông thường cần n^2 phép nhân và n phép cộng $\Rightarrow$ Tốn $O(n^2)$ thời gian.

- Sử dụng kỹ thuật "Divide and conquer":

+ Chia mỗi số nguyên X và Y thành các số nguyên có n/2 chữ số

$$X = A.10^{\frac{n}{2}} + B$$
$$Y = C.10^{\frac{n}{2}} + D$$

trong đó A, B, C, D là các số nguyên có n/2 chữ số.

(Ví dụ X=1234 thì A=12 và B= 34 vì X= $12.10^2 + 34$)

$\Rightarrow$ $X . Y = A.C.10^n + (AD + BC).10^{n/2} + BD$ (1)

+ Với mỗi số có n/2 chữ số, tiếp tục phân tích theo cách trên, quá trình phân tích sẽ dẫn đến một bài toán cơ sở là nhân các số nguyên chỉ gồm một chữ số mà ta dễ dàng thực hiện. Việc tổng hợp kết quả là việc thực hiện các phép toán theo công thức (1).

+ Theo (1) phải thực hiện 4 phép nhân các số nguyên n/2 chữ số (AC, AD, BC, BD), tổng hợp kết quả bằng 3 phép cộng các số nguyên n chữ số và 2 phép nhân với 10^n và $10^{n/2}$. Phép cộng các số nguyên n chữ số chỉ cần O(n), phép nhân với 10^n có thể thực hiện bằng cách thêm vào n chữ số 0 $\Rightarrow$ O(n).

Gọi T(n) là thời gian nhân 2 số nguyên, mỗi số có n chữ số. Ta có phương trình đệ quy:

$$T(n) = \begin{cases} 1 & neu\ n = 1 \\ 4T(\frac{n}{2}) + Cn & neu\ n > 1 \end{cases}$$

Giải T(n) được $T(n) = O(n^2)$ $\Rightarrow$ Chưa cải tiến hơn so với giải thuật nhân 2 số thông thường.

Do đó: + Viết (1) thành dạng

$$XY = AC10^n + [(A-B)(D-C) + AC + BD].10^{n/2} + BD \quad (3)$$

(3) chỉ cần 3 phép nhân các số nguyên n/2 chữ số AC, BD và (A-B)(D-C), 6 phép cộng trừ và 2 phép nhân với 10^n $\Rightarrow$ lấy O(n) $\Rightarrow$ phương trình đệ quy

$$T(n) = \begin{cases} 1 & \textit{nếu } n = 1 \\ 3T(\frac{n}{2}) + Cn & \textit{nếu } n > 1 \end{cases}$$

Giải phương trình đệ quy được $T(n) = O(n^{\log 3}) = O(n^{1.59})$ $\Rightarrow$ Giải thuật này cải thiện hơn rất nhiều.

Function Mult(x,y: integer; n: integer) : integer;

Var m_1,m_2,m_3,A,B,C,D : integer;

dấu : integer; *{lưu trữ dấu của tích xy}*

```
Begin
      dấu :=sign(x)*sign(y);
     x:=ABS(x); y:=ABS(y)
      if n=1 then Mult:=x*y*dấu
      else
        Begin
              A:=Left(x, n div 2);    {A: số nguyên có n/2 chữ số bên trái}
              B:=Right(x, n div 2);
              C:=Left(y, n div 2);
              D:=Right(y, n div 2);
              m1:=Mult(A,C,n div 2);
              m2:=Mult(A-B,D-C,n div 2);
              m3:=Mult(B,D,n div 2);
              Mult:=dấu*[(m1*10n)+(m1+m2+m3)*10n div 2+m3];
        End;
End;
```

$$sign(x) = \begin{cases} 1 \text{ nếu } x>0 \\ -1 \text{ nếu } x<0 \end{cases}$$

c)Xếp lịch thi đấu thể thao

Kỹ thuật "Divide and conquer" không chỉ có ứng dụng trong thiết kế giải thuật mà còn trong nhiều ứng dụng khác của cuộc sống như : xếp lịch thi đấu thể thao theo thể thức đấu vòng tròn một lượt cho n cầu thủ. Mỗi cầu thủ phải đấu với các cầu thủ khác và mỗi cầu thủ chỉ đấu nhiều nhất 1 trận một ngày.

Yêu cầu : Xếp 1 lịch thi đấu sao cho số ngày thi đấu là ít nhất.

- Dễ thấy, tổng số trận đấu của toàn giải là $\frac{n(n-1)}{2}$

$\Rightarrow$ nếu n chẵn $\Rightarrow$ Sắp n/2 cặp thi đấu một ngày $\Rightarrow$ cần ít nhất n-1 ngày.

nếu n lẻ $\Rightarrow$ n-1 là chẵn $\Rightarrow$ sắp (n-1)/2 cặp thi đấu trong một ngày $\Rightarrow$ cần n ngày. Giả sử $n=2^k$ thì n chẵn $\Rightarrow$ cần tối thiểu n-1 ngày.

- Lịch thi đấu là một bảng n dòng, n-1 cột, dòng i biểu diễn cầu thủ i, cột j biểu diễn ngày thi đấu j, ô (i,j) thể hiện cầu thủ phải thi đấu với cầu thủ i trong ngày j.

- Xây dựng lịch thi đấu theo kỹ thuật "Divide and conquer" như sau : Để sắp lịch cho n cầu thủ, ta sắp lịch cho n/2 cầu thủ; để sắp lịch cho n/2 cầu thủ, ta sắp lịch cho n/4

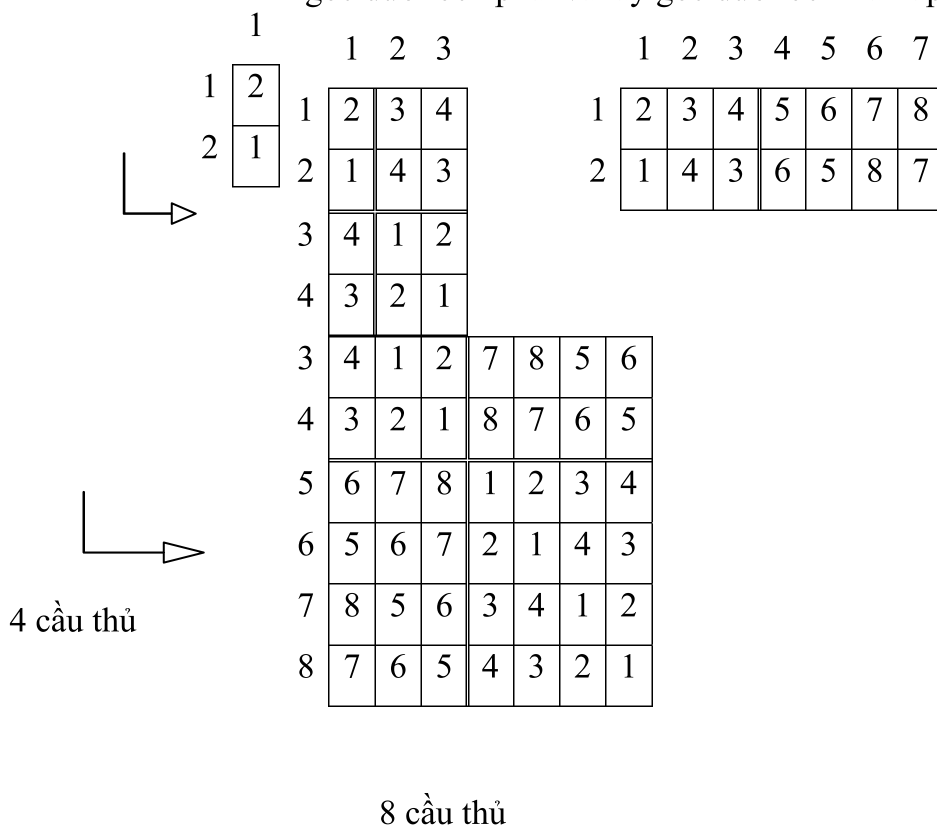
cầu thủ...dẫn đến bài toán cơ sở là sắp lịch thi đấu cho 2 cầu thủ, 2 cầu thủ này sẽ thi đấu 1 trận trong một ngày.

- Tổng hợp lịch thi đấu cho 4 cầu thủ, 8 cầu thủ,...từ lịch thi đấu của 2 cầu thủ như sau:

Xuất phát từ lịch thi đấu của 2 cầu thủ⇒lịch thi đấu cho 4 cầu thủ là một bảng 4 dòng, 3 cột. Lịch thi đấu cho 2 cầu thủ 1 và 2 trong ngày 1 chính là lịch thi đấu cho 2 cầu thủ (bài toán cơ sở)⇒ô (1,1)=”2” ; ô (2,1)=”1”, tương tự ta có lịch thi đấu cho 2 cầu thủ 3 và 4 trong ngày 1⇒ô (3,1)=”4”, ô (4,1)=”3”.

Nhận xét: ô (3,1)=ô (1,1)+2 và ô (4,1)=ô (2,1)+2

Để hoàn thành lịch thi đấu cho 4 cầu thủ ta lấy góc trên bên trái của bảng lắp vào cho góc dưới bên phải và lấy góc dưới bên trái lắp cho góc trên bên phải…



	1
1	2
2	1

	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

4 cầu thủ

	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	6	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

8 cầu thủ

Với kỹ thuật “chia để trị” nói chung sẽ tốt hơn nếu ta chia bài toán cần giải thành các bài toán con có kích thước gần bằng nhau

Ví dụ: Mergesort phân chia bài toán thành 2 bài toán con có cùng kích thước n/2 => Thời gian thực hiện là O(nlogn). Ngược lại trong trường hợp xấu nhất của QuickSort, khi mảng bị phân hoạch lệnh thì thời gian là $O(n^2)$.

Nguyên tắc chung là tìm cách phân chia bài toán thành các bài toán con có kích thước xấp xỉ bằng nhau thì hiệu quả sẽ cao hơn => Gọi là bài toán con cân bằng*(Balancing Subproblems).*

3.2. Quy hoạch động *(Dynamic)*

3.2.1. Nội dung

Tư tưởng chủ đạo của phương pháp quy hoạch động dựa trên nguyên lí tối ưu của Bellman : "*Nếu một dãy các lựa chọn là tối ưu thì mọi dãy con của nó cũng tối ưu*".

Ta biết kĩ thuật chia để trị thường dẫn tới một giải thuật đệ quy. Trong các giải thuật đó, có thể có một số giải thuật thời gian mũ. Tuy nhiên, thường chỉ có một số đa thức các bài toán con, điều đó có nghĩa là ta phải giải một số bài toán con nào đó trong nhiều lần. Thuật toán nhận được sẽ kém hiệu quả. Vì vậy, để tránh việc giải dư thừa một số bài toán con, ta dùng kỹ thuật từ dưới lên (bottom – up) để tạo ra một bảng lưu tất cả các kết quả của các bài toán con và khi cần chỉ cần tham khảo tới kết quả đó được lưu trong bảng mà không phải giải lại bài toán đó. Lấp đầy bảng kết quả các bài toán con theo một quy luật nào đó để nhận được kết quả của bài toán ban đầu (cũng đã được lưu trong một ô nào đó của bảng) được gọi là quy hoạch động.

✓Các thao tác tổng quát của quy hoạch động:

- Tìm nghiệm của các bài toán con (*các trường hợp riêng*) đơn giản nhất
- Xây dựng hàm quy hoạch động (*công thức hoặc quy tắc xây dựng nghiệm của bài toán con thông qua nghiệm của các bài toán con cỡ nhỏ hơn)*
- Lập bảng lưu lại các giá trị của hàm.
- Dùng bảng lưu để truy xuất lời giải tối ưu (*tìm nghiệm của bài toán).*

✓Hạn chế của quy hoạch động:

Phương pháp quy hoạch động không hiệu quả trong các tình huống sau:

- Sự kết hợp lời giải của các bài toán con chưa chắc đã cho ta lời giải của các bài toán lớn hơn.
- Số lượng các bài toán con cần giải quyết và lưu trữ kết quả có thể rất lớn, không thể chấp nhận được.

3.2.2. Ví dụ áp dụng

a) Bài toán tính số tổ hợp

Ta biết công thức tính số tổ hợp chập k của n là

$C_n^k = 1$ nếu k = 0 hoặc k = n

$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$ nếu 0<k<n

° *Giải thuật đệ quy*

```
Function Tohop(k,n : Integer): integer;
Begin
    If (k=0) or (k=n) then Tohop:=1
    Else Tohop:=Tohop(k-1,n-1)+Tohop(k,n-1);
End;
```

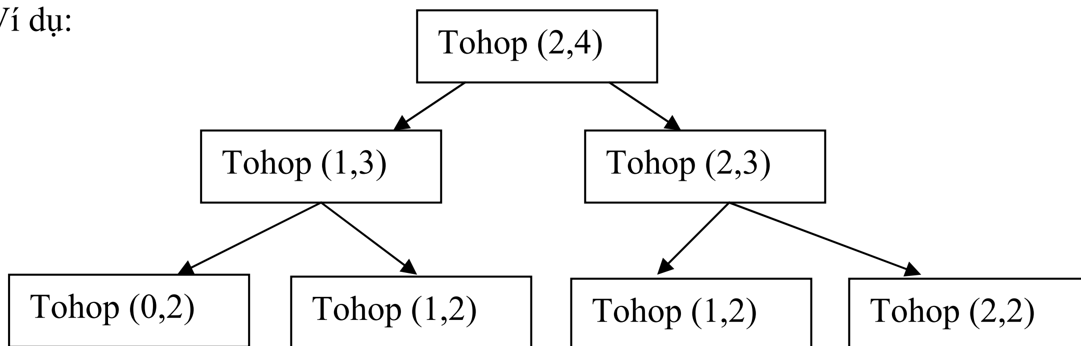
- *Độ phức tạp tính toán*: Gọi T(n) là thời gian tính C_n^k , Ta có phương trình đệ quy

$$T(n) = \begin{cases} C_1 & \text{nếu } k=0 \text{ hoặc } k=n \\ 2T(n-1) + C_2 & \text{nếu } 0<k<n \end{cases}$$

Giải phương trình đệ quy ta được $T(n) \sim O(2^n)$.

Như vậy giải thuật đệ quy trên có thời gian thực hiện là hàm mũ trong khi chỉ có một đa thức các bài toán con, chứng tỏ có những bài toán con được giải nhiều lần.

Ví dụ:



Như vậy để tính Tohop(2,4) phải tính Tohop(1,2) 2 lần… Quy hoạch động sẽ khắc phục tình trạng trên bằng cách xây dựng một bảng gồm n+1 dòng (từ 0..n) và n+1 cột từ (0..n) và điền giá trị ô O(i,j) theo qui tắc tam giác Pascal:

j / i	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

O(0,0)=1; O(i,0)=1;

O(i,i)=1 với $0<i\leq n$

O(i,j)=O(i-1,j-1) + O(i-1,j) với $0 <j<i \quad \forall n$

Giá trị ở ô O(n,k) chính là Tohop(n,k);

```
Function  TOHOP(n,k : integer) : integer;
Var  C: array[0..n,0..n] of integer;
        i,j: integer;
Begin
        C[0,0]:=1;
        For i:=1 to n do
        Begin
            C[i,0]:=1; C[i;i]:=1;
            For j:= 1 to i-1 do C[i,j]:=C[i-1,j-1]+C[i-1,j];
        End;
        TOHOP:=Tohop[n,k];
End;
```

Độ phức tạp: $T(n) = O(n^2)$.

b) Xâu con chung

Bài toán: Cho hai xâu ký tự x và y, hãy tìm xâu lý tự c là xâu con chung của x và y và c có độ dài lớn nhất có thể được.

Ví dụ: x = ab132sc và y = 1abdc thì c = abc

Nhận xét: Nếu x và y có độ dài đủ nhỏ thì có thể giải bài toán bằng cách duyệt mọi xâu con chung và lưu lại xâu có độ dài lớn nhất. Tuy nhiên cách làm này không thể

đáp ứng về mặt thời gian khi x và y có độ dài lớn. Vì vậy ta sẽ dùng phương pháp quy hoạch động để giải quyết bài toán trên như sau:

Gọi m và n lần lượt là độ dài của các xâu x và y.

- Nếu một trong hai số m = 0 hoặc n = 0 thì c là xâu rỗng.
- Xét các đoạn đầu của 2 xâu x và y có độ dài i và j tương ứng ($x_1, x_2, .., x_i$) và ($y_1, y_2, .., y_j$) với $0 \le i \le m, 0 \le j \le n$. Gọi L(i,j) là độ dài lớn nhất xâu con chung của hai xâu này. Khi đó L(m,n) sẽ là độ dài lớn nhất của hai xâu x và y.

Ta có cách tính L(i,j) thông qua L(s,t) với $s \le i, t \le j$ theo quy tắc sau:

+ nếu i=0 hoặc j=0 thì L(i,j)=0.

+ nếu i >0 và j >0 và $x_i \ne y_j$ thì L(i,j) = Max{L(i,j-1), L(i-1,j)}

+ nếu i >0 và j >0 và $x_i = y_j$ thì L(i,j)=1+L(i-1,j-1).

- Lưu các giá trị L(i,j) vào mảng L[0..m, 0..n]. Từ quy tắc trên ta thấy nếu biết L[i,j-1], L[i-1,j], L[i-1,j-1] ta tính ngay được L[i,j] $\Rightarrow$ Có thể tính được các phần tử của mảng L[0..m,0..n] từ góc trên trái lần lượt theo các đường chéo song song.

```
Procedure  Xau_con_chung;
Var  x, y, c:String;
     I, j: Byte;
     L: array[0..250,0..250] of byte;
Begin
     For i:= 1 to length(x) do L[0,i]:=0;
     For j:= 0 to length(y) do L[j,0]:=0;
     For i:=1 to length(x) do
     For j:=1 to length(y) do
        If x[i] = y[j] then L[i,j]:= L[i-1,j-1] + 1
        Else   If L[i-1,j] > L[i,j-1] then L[i,j]:= L[i-1,j]
               Else L[i,j]:= L[i,j-1]
End;
```

*Vấn đề truy xuất kết quả:

Từ mảng L đã được làm đầy, ta xây dựng xâu con chung dài nhất có độ dài là k = L[m,n]. Ta xác định các thành phần của c = ($c_1, c_2, .., c_k$) lần lượt từ bên phải. Trong bảng L xuất phát từ ô L[m,n], đặt c =' ' , giả sử đang ở ô L[i,j] :

- Nếu $x_i = y_j$ thì gán c = x_i + c, và đi lên ô L[i-1,j-1].
- Nếu $x_i \ne y_j$ thì :

+ hoặc L[i,j] =L[i,j-1] thì đi tới ô L[i,j-1]

+ hoặc L[i,j] =L[i-1,j] thì đi tới ô L[i-1,j].

Quá trình cứ tiếp tục khi ta xác định được tất cả các thành phần của c (nghĩa là khi gặp ô L[i,j] nào đó có giá trị 0).

```
Procedure   Truy_vet;
Var  i,j: byte;
Begin
     C:=’’; i:=length(x); j:=length(y);
     While  L[i,j]<>0 do
          If  x[i] = y[j] then
              Begin
                   C:=x[i]+c; dec(i); dec(j);
              End
          Else   If  L[i,j] = L[i-1,j] then dec(j)
                   Else dec(i);
      Return(c);
End;
```

*Ví dụ : Xây dựng bảng L[6,7]

x = abd4eb

y = 1ab4cde

y \ x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1
3	0	1	2	2	2	2	2
4	0	1	2	2	3	3	3
5	0	1	2	2	3	3	3
6	0	1	2	3	3	3	3
7	0	1	2	3	3	4	4

C = ‘a b 4 e’

Chú ý:

- Trên bảng lưu, khi truy vết có thể có nhiều xâu con chung có độ dài lớn nhất.
- Sử dụng thuật toán trên kết hợp với cấu trúc dữ liệu kiểu con trỏ ta có thể mở rộng bài toán cho trường hợp tìm dãy con chung của hai dãy số nguyên có độ dài lên tới hàng ngàn đơn vị.

3.3. Phương pháp tham lam (*Greedy Method*)

3.3.1. Bài toán tối ưu tổ hợp

Có dạng tổng quát:

- $f(x) = \sum_{i=1}^{n} C_i x_i$ xác định trên một tập hữu hạn các phần tử D. Hàm f(x) được gọi là hàm mục tiêu.
- Mỗi phần tử $X \in D$ có dạng $X=(x_1,x_2, ..x_n)$ được gọi là một phương án.
- Cần tìm một phương án $X \in D$ sao cho hàm f(x) đạt Min(max). Phương án X như thế gọi là phương án tối ưu.

Có thể tìm phương án tối ưu bằng phương pháp "vét cạn" nghĩa là xét tất cả các phương án trong tập D (hữu hạn) để xác định phương án tốt nhất. Mặc dù D là hữu hạn nhưng để tìm một phương án tối ưu cho một bài toán kích thước n bằng phương pháp "vét cạn" ta có thể sẽ tốn thời gian hàm mũ. Với nhiều bài toán tối ưu hoá, quả là quá thừa khi dùng quy hoạch động để xác định các phương án tối ưu, thay vì thế ta có thể dùng các thuật toán đơn giản và hiệu quả hơn.

Phương pháp Greedy Method sẽ giải bài toán tối ưu tổ hợp mà thời gian có thể chấp nhận được, tuy nhiên có thể chỉ đạt được phương án tốt chứ không phải là tối ưu.

3.3.2. Nội dung

Khác với quy hoạch động, thường giải quyết các bài toán con từ dưới lên, một chiến lược tham lam thường tiến triển theo cách từ trên xuống, phương án X được xây dựng bằng cách lựa chọn từng thành phần x_i của X cho đến khi hoàn chỉnh (đủ n thành phần). Với mỗi x_i, ta sẽ chọn x_i tối ưu. Với cách này thì có thể ở bước cuối cùng ta không còn gì để chọn mà phải chấp nhận một giá trị cuối cùng còn lại.

Lược đồ của phương pháp tham lam:

Procedure Greedy_Method(A,X);

{Xây dựng phương án X từ tập các đối tượng A}

Begin

X:= ϕ;

While A <> ϕ **do**

```
    Begin
        x:= Select(A); {Hàm chọn x tốt nhất trong A};
        A:=A-{x};
        If   X∪{x} chấp nhận được   then   X:=X∪{x};
    End;
    Return (X);        {phương án tối ưu}
End;
```

Ta có thể dễ dàng thấy tại sao các thuật toán như thế được gọi là "*tham lam*". Tại mỗi bước, nó chọn "*miếng ngon nhất*" (được xác định bởi hàm chọn), nếu thấy có thể nuốt được (có thể đưa vào nghiệm) nó sẽ xơi ngay, nếu không nó sẽ bỏ đi, sau này không bao giờ xem xét lại.

Cần lưu ý rằng, thuật toán tham lam trong một số bài toán, nếu xây dựng được hàm chọn thích hợp có thể cho nghiệm tối ưu. Trong nhiều bài toán, thuật toán tham lam chỉ tìm được nghiệm gần đúng với nghiệm tối ưu.

3.3.3.Các ví dụ áp dụng:

a)Bài toán đổi tiền

Input : - Có n loại tiền, mỗi loại tiền có giá trị tương ứng là $d_1,d_2,..,d_n$.

- Lượng tiền M đồng.

Output : Đổi M đồng ra tiền lẻ sao cho số loại tiền đổi là ít nhất.

*Phân tích : Cần phải tìm 1 nghiệm $X = (x_1,x_2,..x_m)$ với x_i là số loại tiền thứ i có giá trị d_i sao cho $M=\sum_{i=1}^{m} x_i d_i \Rightarrow$ Tìm cách đổi sao cho tổng loại tiền cần đổi là ít nhất. Vậy ta sẽ bắt đầu đổi từ đồng xu có giá trị lớn nhất và cứ giảm dần cho đến khi số tiền M đã được đổi hết thì thông báo là tìm được nghiệm, ngược lại thì thông báo không đổi được.

Áp dụng tư tưởng của thuật toán tham lam , ta có thể viết như sau:

```
Function  Đổitiền_Thamlam(M);
Const  D={d1,d2,..,dn}              { mảng lưu giá trị từng loại tiền}
Var  x, sum, i;                     {x là nghiệm bài toán}
Begin
    - Sắp xếp D giảm dần
    X=ϕ ; sum:=ϕ ; i:=1;
    While (sum≠M) and (i <= n) do  {Trong khi chưa đổi hết tiền}
```

```
        Begin
            xi:= (M - sum) div di ;  {Số tờ tiền loại di}
            if (sum+xi*di ≤ M) then
            Begin
              X:=X+{xi};
              sum:=sum+xi*di;  {Số tiền đã đổi được tính đến loại tiền i}
            End;
            i:=i+1;
        end;
        If sum=M then Return(X) {Trả lại nghiệm của bài toán}
        Else  <Thông báo không đổi được>;
End;
```

Nhận xét: - Tính chất tham lam thể hiện ở chỗ, tại mỗi bước luôn chọn "miếng ăn ngon nhất" mà không để ý hậu quả sau này. Cho nên, với một số trường hợp thuật toán cho ta nghiệm tối ưu, nhưng trong nhiều trường hợp nghiệm tìm được không phải là nghiệm tối ưu mà chỉ gần đúng với nghiệm tối ưu, thậm chí còn không tính được nghiệm đúng. Ví dụ: Nếu M=13 và các loại tiền có mệnh giá là: $d_1=3$, $d_2=4$, $d_3=6$. Cần tìm cách đổi 13 đồng sao cho số tiền đổi là ít nhất. Với thuật toán trên ta cần 2 tờ 6 đồng dư 1 đồng. Như vậy số tiền này sẽ không đổi được, nhưng trong thực tế, ta có thể đổi được dễ dàng với 1 tờ 6 đồng, 1 tờ 4 đồng và một tờ 3 đồng.

- Tuy nhiên ta hoàn toàn có thể sử dụng phương pháp tham lam để đạt nghiệm tối ưu với một số bài toán như : Thuật toán Kruskal tìm cây khung nhỏ nhất,…

b) Bài toán đường đi của người giao hàng *(TSP - Traveling Salesman Problem)*

Một người giao hàng cần đi giao hàng tại n thành phố. Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, khoảng cách từ 1 thành phố tới các thành phố khác là xác định được (địa lí, thời gian, cước phí) độ dài. Hãy lập một chu trình sao cho tổng độ dài các cạnh là nhỏ nhất.

- Với phương pháp "vét cạn", ta xét tất cả các chu trình, mỗi chu trình tính tổng độ dài các cạnh rồi chọn một chu trình có tổng độ dài nhỏ nhất, do đó cần xét tất cả là $\frac{(n-1)!}{2}$ chu trình $\Rightarrow$ Độ phức tạp O(n!). Vì vậy, ta có thể sử dụng phương pháp tham lam sẽ

cho phương án tốt (không phải phương án tối ưu) nhưng chỉ tốn thời gian đa thức như sau:

<u>*Thuật toán TSP_Greedy:</u>

① Tính độ dài của tất cả các cạnh (có tất cả n(n-1)/2 cạnh).

② Xét các cạnh có độ dài từ nhỏ đến lớn để đưa vào chu trình.

③ Một cạnh sẽ được đưa vào chu trình nếu cạnh đó thoả mãn hai điều kiện sau:

- Không tạo thành một chu trình thiếu(Không đi qua đủ n đỉnh).
- Không tạo thành một đỉnh có cấp ≥ 3(tức là không được có nhiều hơn hai cạnh xuất phát từ một đỉnh, vì mỗi thành phố chỉ được đến một lần).

④ Lặp lại bước 3 cho đến khi xây dựng được một chu trình.

Với phương pháp này ta chỉ cần n(n-1)/2 phép chọn nên ta có một giải thuật cần $O(n^2)$ thời gian.

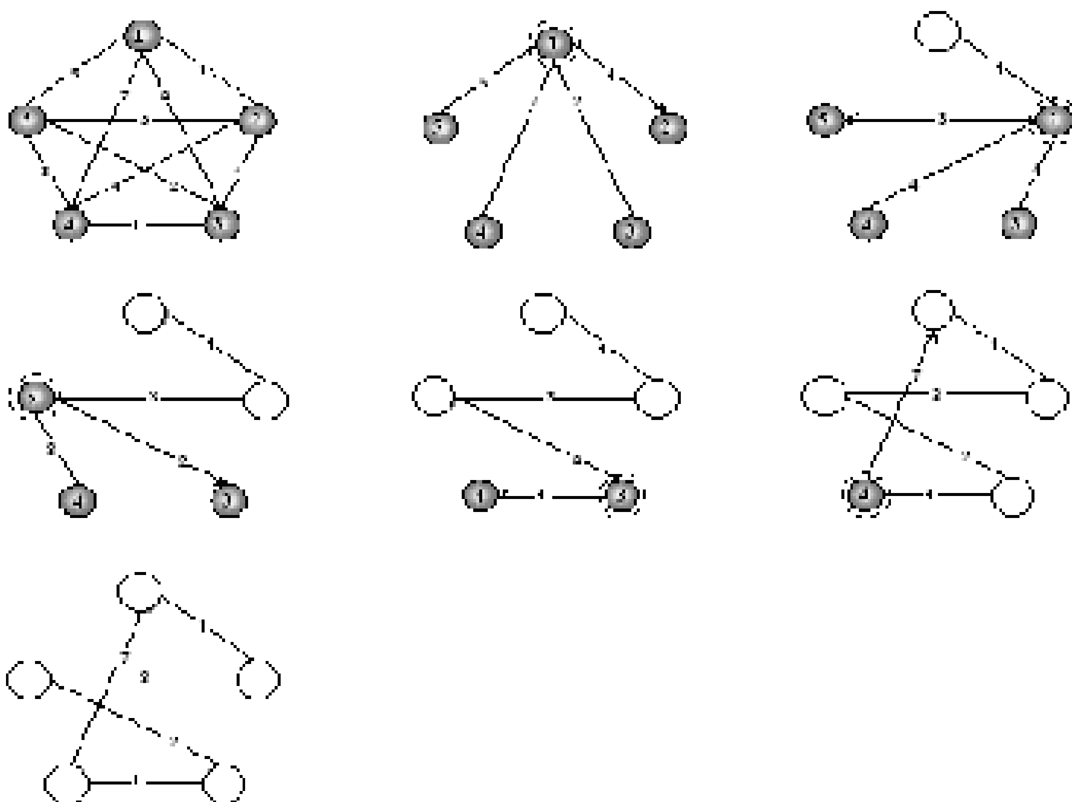
Một cách tiếp cận khác của kỹ thuật tham lam vào bài toán này là:

① Xuất phát từ một đỉnh bất kỳ, chọn một cạnh có độ dài nhỏ nhất trong tất cả các cạnh đi ra từ đỉnh đó để đến đỉnh kế tiếp.

② Từ đỉnh kế tiếp ta lại chọn một cạnh có độ dài nhỏ nhất đi ra từ đỉnh này thoả mãn hai điều kiện nói trên để đi tới đỉnh kế tiếp.

③ Lặp lại bước 2 cho đến khi đi tới đỉnh n thì quay trở về đỉnh xuất phát.

Ví dụ : Cho bài toán TSP 5 đỉnh được biểu diễn như một đồ thị G = (V, E), và quá trình thực hiện giải bài toán bằng thuật toán tham lam thể hiện trong hình sau:



Hình : Quá trình tìm hành trình theo nguyên lý Greedy, đỉnh số 1 là đỉnh xuất phát

Kết quả: Hành trình tìm được có chiều dài 14 (*Hành trình tối ưu là 13).* Độ phức tạp $O(n^2)$.

c) Bài toán xếp ba lô *(xếp balô giá trị nguyên)*

- Input: Một ba lô có thể tích B, n đồ vật có thể tích: $a_1, a_2, \ldots a_n$,

  Giá trị tương ứng của các đồ vật là: $p_1, p_2, \ldots p_n$

  Số lượng mỗi loại đồ vật là không hạn chế, x_i nguyên là số lượng loại đồ vật i.

- Output: Tìm nhóm đồ vật thoả mãn $\sum_{i=1}^{n} a_i x_i \le B$ và $\sum_{i=1}^{n} p_i x_i$ đạt max ?

Theo yêu cầu của bài toán ta cần những đồ vật có giá trị cao mà thể tích nhỏ để có thể mang được nhiều "đồ quý". Vì vậy ta quan tâm tới đơn giá của từng loại đồ vật tức là tỷ lệ giá trị/ thể tích, đơn giá càng cao thì đồ càng quý. Khi đó phương pháp tham lam được áp dụng giải bài toán trên trong thời gian đa thức như sau:

① Tính đơn giá cho các loại đồ vật.

② Xét các đồ vật theo thứ tự đơn giá từ lớn đến nhỏ.

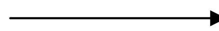
③ Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà thể tích còn lại của balô cho phép.

④ Xác định thể tích còn lại của balô và quay lại bước 3 cho đến khi không còn chọn được đồ vật nào nữa.

- Độ phức tạp tính toán: $O(n^2)$.

*Ví dụ: Cho balô có thể tích B=37 và 4 loại đồ vật có trọng lượng và giá trị tương ứng như sau:

Loại đồ vật	Thể tích a_i	Giá trị p_i
A	15	30
B	10	25
C	2	2
D	4	6



Đồ vật	a_i	p_i	Đơn giá p_i/a_i
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

- Chọn B: 3 cái $\Rightarrow$ Thể tích còn lại 37-3*10=7
- Chọn A : a_i=15 >7 $\Rightarrow$ không được.
- Chọn D : 1 cái $\Rightarrow$ Thể tích còn lại : 7-1*4=3
- Chọn C : 1 cái $\Rightarrow$ Thể tích còn lại: 3-1*2=1

$\Rightarrow$ Tổng thể tích : 3*10+1*4+1*2=36

Tổng giá trị : 3*25+1*6+1*2=83.

3.4. Phương pháp nhánh cận *(Branch- and- Bound)*

3.4.1. Nội dung

Với các bài toán tìm phương án tối ưu, nếu ta xét hết tất cả các phương án thì mất nhiều thời gian, nhưng nếu sử dụng phương pháp Greedy thì phương án tìm được chưa chắc đã là phương án tối ưu. Phương pháp nhánh cận là một dạng cải tiến của phương pháp quay lui được áp dụng để tìm nghiệm của bài toán tối ưu.

Với phương pháp nhánh cận ta sẽ đi xây dựng cây tìm kiếm phương án tối ưu, nhưng không xây dựng toàn bộ cây mà sử dụng giá trị cận để hạn chế bớt các nhánh.

- Cây tìm kiếm phương án có nút gốc biểu diễn cho tập tất cả các phương án có thể có, mỗi nút lá biểu diễn một phương án nào đó. Nút 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 ta xác định được một giá trị cận (là giá trị gần với giá của các phương án).

  + Với bài toán tìm Min ta xác định cận dưới $\le$ giá của phương án.

  + Với bài toán tìm Max ta xác định cận trên $\ge$ giá của phương án.

Như vậy có thể nói: Phương pháp Nhánh cận là một dạng cải tiến của phương pháp "vét cạn", ta có lược đồ của phương pháp nhánh cận như sau:

Procedure TRY(i: integer);

Var j: integer;

Begin

For i $\in$ Tập giá trị đề cử **do**

If <chấp nhậnj> **then**

Begin

x_i:=j;

if i=n then <ghi nhận>

else if g(x_1,x_2,..,x_n) $\le$ cận dưới tạm thời **then** TRY (i+1)

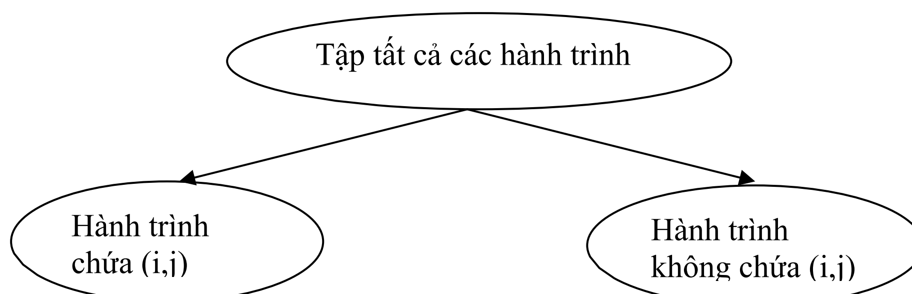
End;

End;

3.4.2. Các bài toán áp dụng

a)Bài toán đường đi của người giao hàng (TSP)

<u>*Phân nhánh:</u> Trong bài toán này khi tìm kiếm lời giải, chúng ta sẽ phân tập các hành trình ra thành hai tập con : một tập gồm những hành trình chứa một cạnh (i,j) nào đó cũn tập kia gồm những hành trỡnh khụng chứa cạnh này:



Khi đó ta có thể xây dựng cây tìm kiếm phương án là cây nhị phân, trong đó :

- Nút gốc là nút biểu diễn cho cấu hình bao gồm tất cả các hành trình có thể có.

- Mỗi nút có hai con, con trái biểu diễn cấu hình bao gồm tất cả các hành trình chứa một cạnh nào đó, nút con phải biểu diễn cấu hình bao gồm tất cả các hành trình không chứa cạnh đó (các cạnh xét phân nhánh được xếp theo một thứ tự nào đó, chẳng hạn thứ tự từ điển).
- Mỗi nút sẽ kế thừa các thuộc tính của tổ tiên của nó và có thêm một thuộc tính mới(chứa hay không chứa một cạnh nào đó).
- Nút lá biểu diễn cho 1 cấu hình chỉ bao gồm 1 phương án.
- Trong quá trình phân nhánh phải đảm bảo các điều kiện ràng buộc của bài toán ⇒ tại mỗi nút ta cần quy định trên nguyên tắc là mọi đỉnh trên chu trình đều có cấp 2 và không tạo ra một chu trình thiếu.

Tính cận dưới: Đây là bài toán tìm Min nên ta sử dụng cận dưới. Cận dưới tai mỗi nút là số nhỏ hơn hoặc bằng giá của tất cả các phương án biểu diễn bởi nút đó. Giá của một phương án là tổng độ dài 1 chu trình.

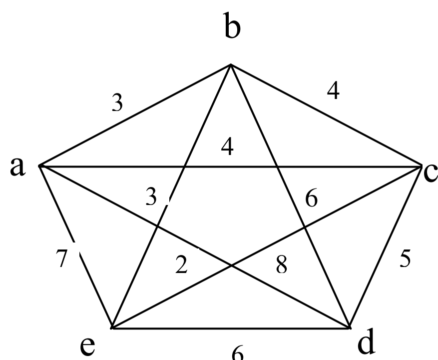
- Tính cận dưới cho nút gốc : mỗi đỉnh chọn hai cạnh có độ dài nhỏ nhất. Cận dưới của nút gốc bằng tổng độ dài tất cả các cạnh được chọn chia cho 2.
- Các nút khác : Lựa chọn 2 cạnh có độ dài nhỏ nhất thỏa mãn điều kiện ràng buộc (chứa cạnh này,không chứa cạnh kia).

Kỹ thuật nhánh cân: Kết hợp hai kỹ thuật trên để xây dựng cây tìm kiếm phương án theo quy tắc sau:

- Xây dựng nút gốc, bao gồm tất cả các phương án, tính cận dưới cho nút gốc.
- Phân nhánh cho mỗi nút, tính cận dưới cho cả 2 con.
- Nếu cận dưới của một nút con lớn hơn hoặc bằng giá nhỏ nhất tạm thời của một phương án đã được tìm thấy thì không cần xây dựng các nhánh con cho nút này nữa (cắt tỉa).
- Nếu cả hai con đều có cận dưới nhỏ hơn giá nhỏ nhất tạm thời của một phương án đã được tìm thấy thì nút nào có cận dưới nhỏ hơn sẽ được ưu tiên phân nhánh trước.
- Mỗi lần quay lui để xét nút con chưa được xét của một nút ta phải xem xét lại nút con đó để có thể cắt tỉa các nhánh của nó hay không, vì có thể một phương án có giá trị nhỏ nhất tạm thời chưa được tìm thấy.
- Sau 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 dược là phương án tối ưu.

Trong quá trình xây dựng cây có thể ta đã xây dựng được một số nút lá (một phương án). Giá nhỏ nhất trong số các giá của các phương án này được gọi là giá nhỏ nhất tạm thời.

*Ví dụ : Xét bài toán TSP có 5 đỉnh:



Các cạnh theo thứ tự từ điển: ab,ac,ad,ae,bc,bd, be, cd,ce,dc.

- Phân nhánh: Xuất phát từ nút gốc chứa tập tất cả các hành trình, nút con trái là hành trình chứa cạnh ab, nút con phải là hành trình không chứa cạnh ab,…
- Cận dưới: Được tính như sau

Tính CD cho nút gốc:

+đỉnh a chọn ab, ad.

+đỉnh b chọn ba,be.

+đỉnh c chọn ca,cb.

+đỉnh d chọn da,dc.

+đỉnh e chọn eb,ed.

$\Rightarrow$ Tổng là 35 $\Rightarrow$ cận dưới của nút gốc A là 35/2=17,5.

Tính cận dưới cho nút D: Điều kiện ràng buộc : chứa cả ab, ac, không chứa ad, ae.

+đỉnh a chọn ab,ac.

+đỉnh b chọn ba,be.

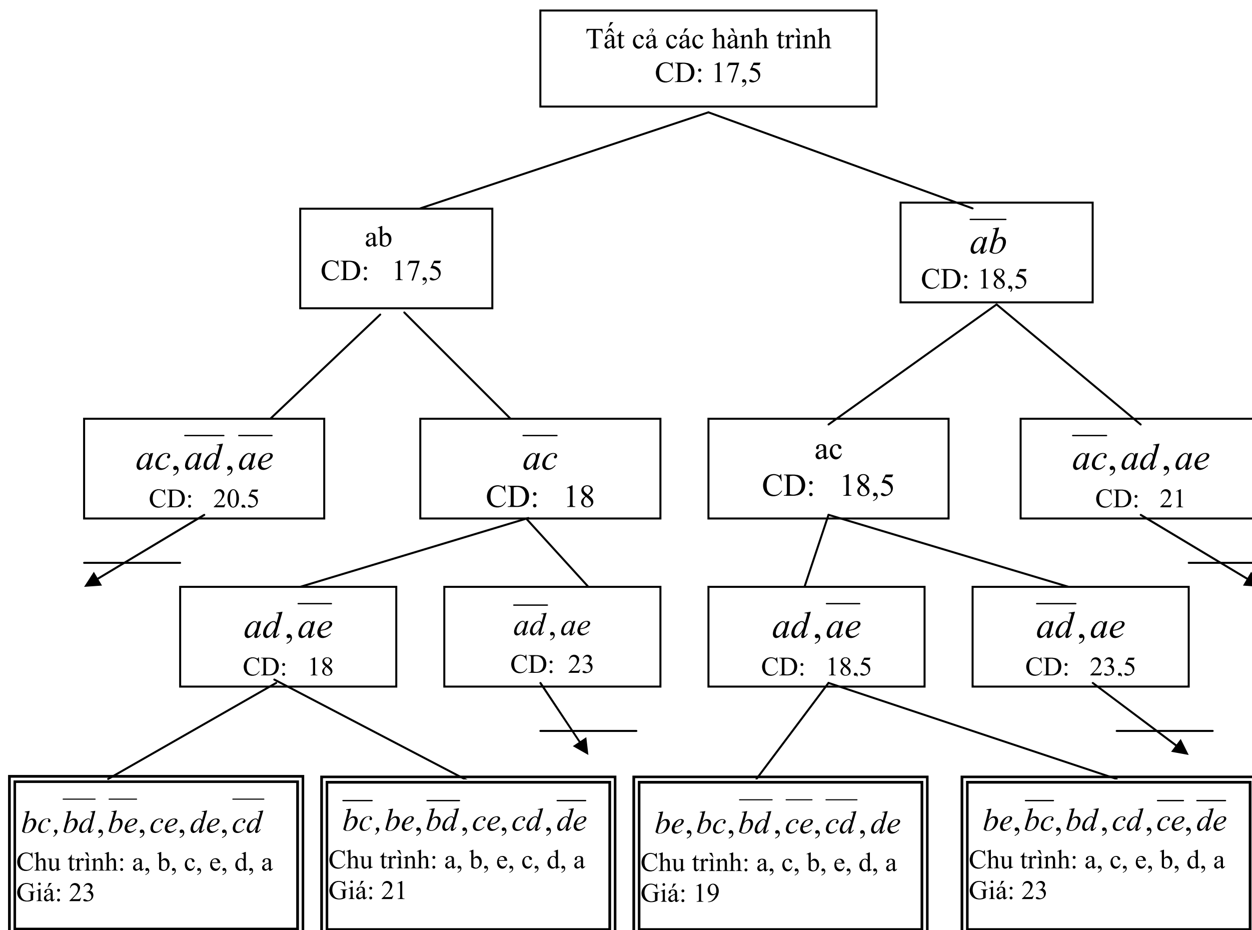
+đỉnh c chọn ca,cb.

+đỉnh d chọn de,dc.

+đỉnh e chọn eb,ed.

$\longrightarrow$ Tổng là 41 $\Rightarrow$ cận dưới của D là 20,5.

tương tự cho các nút còn lại …



Sau khi phân nhánh và cắt tỉa ta tìm được hành trình tối ưu là: a, c, b, e, d, a với chi phí nhỏ nhất 19 (Phương pháp tham lam là 21).

<u>* Phương pháp cài đặt:</u>

- Xây dựng thủ tục rút gọn ma trận chi phí để tính cận dưới.
- Xây dựng thủ tục chọn cạnh phân nhánh.
- Xây dựng thủ tục ngăn cấm tạo thành hành trình con.
- Xây dựng thủ tục đệ quy TSP- nhánh cận tìm hành trình tối ưu.

b)Bài toán xếp ba lô

Ta xét bài toán xếp balô giá trị nguyên đã nêu ở phần trước và phương pháp nhánh cận giải bài toán này:

Tóm tắt bài toán: $\sum_{i=1}^{n} x_i w_i \le B,$ $\sum_{i=1}^{n} x_i p_i \to Max$

<u>Phương pháp Nhánh cận giải bài toán này như sau:</u>

Ta thấy đây là một bài toán tìm Max. Danh sách các đồ vật được sắp xếp theo thứ tự giảm của đơn giá để xét phân nhánh.

1. Nút gốc biểu diễn trạng thái ban đầu của balô(chưa chọn một vật nào). Tổng giá trị TGT=0; cận trên của nút gốc : CT= B*đơn giá Max.

2. Nút gốc có các nút con tương ứng với các khả năng chọn đồ vật theo đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số:

    TGT = TGT(cũ)+(số đồ vật được chọn)*(giá trị mỗi vật)

    B = B(cũ) - (số đồ vật được chọn)*(trọng lượng mỗi vật).

    CT = TGT + B(mới)*(đơn giá vật kế tiếp).

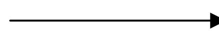
3. Trong các nút con, ta sẽ ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút phải xác định lại các thông số TGT, B, CT .

4. Lặp lại bước 3 với chú ý: với những nút có cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời của một phương án đã tìm thấy thì không cần phân nhánh cho nút đó nữa(cắt tỉa).

5. Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá trị lớn nhất là phương án cần tìm.

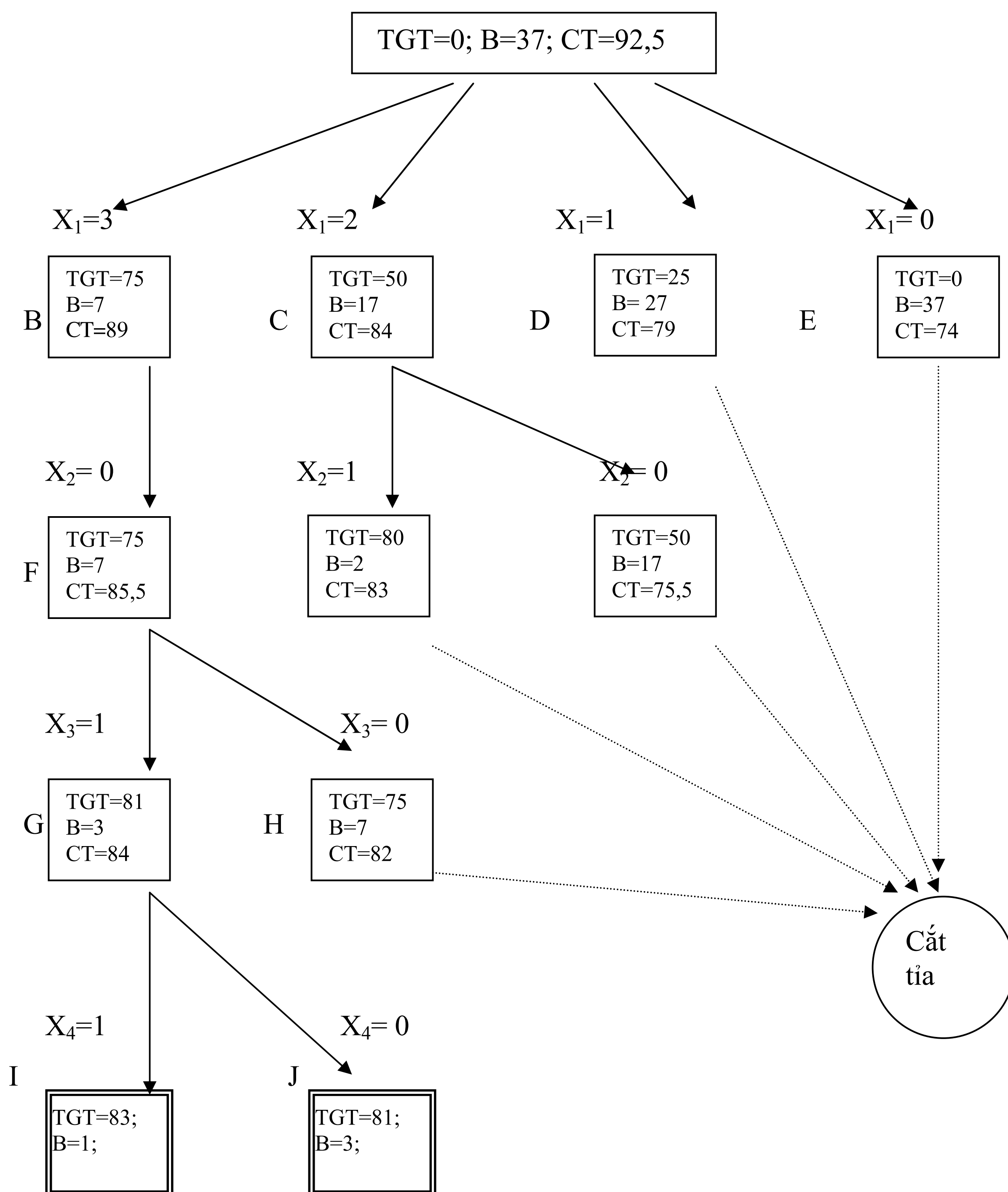
*Ví dụ: Cho balô có thể tích B=37 và 4 loại đồ vật có trọng lượng và giá trị tương ứng như sau:

Loại đồ vật	Thể tích a_i	Giá trị p_i
A	15	30
B	10	25
C	2	2
D	4	6



Đồ vật	a_i	p_i	Đơn giá p_i/a_i
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

Gọi x_1, x_2, x_3, x_4 là số lượng cần chọn tương ứng của các đồ vật B, A, D, C.


TGT=0; B=37; CT=92,5
X1=3
X1=2
X1=1
X1= 0
B
TGT=75
B=7
CT=89
C
TGT=50
B=17
CT=84
D
TGT=25
B= 27
CT=79
E
TGT=0
B=37
CT=74
X2= 0
X2=1
X2= 0
F
TGT=75
B=7
CT=85,5
TGT=80
B=2
CT=83
TGT=50
B=17
CT=75,5
X3=1
X3= 0
G
TGT=81
B=3
CT=84
H
TGT=75
B=7
CT=82
Cắt
tỉa
X4=1
X4= 0
I
J
TGT=83;
B=1;
TGT=81;
B=3;

Chương 4

LÝ THUYẾT LẬP LỊCH

Lịch biểu (Schedule) là một loại kế hoạch để thực hiện một công việc nào đó. Vì vậy việc lập lịch (Scheduling) gặp hàng ngày, ví dụ: lập lịch để các ôtô vào nhà máy sửa chữa, lập lịch để sản xuất các bộ phận máy bay, lập lịch để sắp xếp công việc hàng ngày… Tóm lại, lập lịch là rất cần thiết để các công việc khi xử lí không bị chồng chéo, không bỏ sót. Không những thế, việc lập lịch phải đáp ứng những yêu cầu cao hơn như: nâng cao hiệu quả công việc, nâng cao năng suất lao động, giảm thời gian chờ đợi, giảm chi phí phải trả…Do đó cần phải tìm ra các thuật toán, đánh giá các thuật toán đó để tìm ra những thuật toán tối ưu hoặc gần tối ưu giải quyết các bài toán lập lịch thỏa mãn yêu cầu trên.

Bài toán lập lịch có 2 dạng chính: Cho n công việc xử lý trên máy

- Bài toán lập lịch dạng tĩnh (*Static*): Các công việc và máy đã biết trước.
- Bài toán lập lịch dạng động (*Dynamic):* Các công việc và máy chưa biết trước.

4.1. Vấn đề lập lịch tối ưu

4.1.1. Bài toán

Có n ô tô sẵn sàng trước nhà máy để vào sửa chữa, nhà máy được biết về mỗi ô tô qua 2 tham số:

- Hạn định mà nó phải sửa xong.
- Thời gian cần thiết để sửa chữa ôtô đó.

Nhà máy làm việc không ngừng cho đến khi mọi ô tô sửa xong. Tại mỗi thời điểm, nhà máy chỉ xử lí được một ô tô, sửa xong ôtô này mới sửa ô tô khác.

Hãy sắp xếp các ô tô theo một thứ tự nhất định để theo đó chờ đến lượt sửa chữa, sao cho số lượng ô tô được sửa đúng hạn là nhiều nhất.

Biểu diễn bài toán:

* Input:

- n ô tô, thời điểm có thể bắt đầu xử lí là như nhau:

  $r_i = 0, \forall i = 1, 2, \ldots, n.$
- Hạn định sửa chữa: d_i (due date).
- Thời gian cần thiết để sửa: t_i (time).
- Nhà máy làm việc liên tục.

- Mỗi thời điểm chỉ xử lí được một ô tô.
- Sửa xong ô tô này mới được sửa ô tô khác: không cho phép ngắt.

* Output:

Kí hiệu b_i : thời điểm bắt đầu thực sự xử lí ô tô i.

c_i : thời điểm sửa xong ô tô i.

$\begin{cases} c_i \le d_i : \text{đúng hạn} \\ c_i > d_i : \text{quá hạn.} \end{cases}$

W_i : Tiền phạt

Tìm lịch biểu S sao cho số lượng ô tô được sửa đúng hạn là nhiều nhất (Lượng tiền phạt nhỏ nhất).

* Ví dụ: Cho các tham số dưới dạng bảng sau:

ô tô	Hạn định d_i	Thời gian t_i
x_1	7	3
x_2	5	2
x_3	8	4
x_4	12	4
x_5	17	6
x_6	9	2
x_7	20	3

Nếu sửa chữa theo thứ tự $S = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ thì số lượng ô tô được sửa đúng hạn chỉ có 2. Vì sau khi sửa xong ô tô x_1, x_2 thì ô tô x_3 đã bị quá hạn (do $t_1+t_2+t_3 > d_3$) tiếp theo 4 ô tô sau cũng bị quá hạn.

Nếu sửa chữa theo thứ tự $S = (x_2, x_1, x_6, x_4, x_5, x_3, x_7)$ thì số lượng ô tô được sửa đúng hạn là 5.

4.1.2. Nhận xét

- Bài toán trên thuộc loại lập lịch tối ưu, với chuẩn tối ưu là số lượng ô tô được sửa đúng hạn là nhiều nhất (hay lượng tiền phạt là ít nhất : $\sum_{i=1}^{n} w_i \to \text{Min}$)

- Bài toán luôn giải được vì nó thuộc loại bài toán tổ hợp hữu hạn phần tử nên có thể dùng phương pháp "vét cạn": lấy tất cả các hoán vị của n ô tô, chọn hoán vị nào mà theo thứ tự đó, số lượng ô tô được sửa đúng hạn là nhiều nhất.
- Bài toán trên có thể có nhiều lịch biểu tối ưu, tức là có nhiều thứ tự của n ô tô thỏa mãn chuẩn tối ưu.
- Bài toán trên đã được chứng minh là NPC. Nếu dùng phương pháp "vét cạn" thì sẽ tốn rất nhiều thời gian (*hàm mũ*). Do đó ta cần tìm cách xây dựng được lịch biểu tối ưu trong thời gian đa thức.

4.1.3. Tình hình giải bài toán lập lịch hiện nay

Lý thuyết lập lịch hiện nay đã, đang , và sẽ giải quyết được các vấn đề:

- Phân loại bài toán lập lịch.
- Nghiên cứu cách thức xây dựng lịch biểu tối ưu theo một tiêu chuẩn nào đó và hơn nữa làm sao để tạo ra nó thật nhanh (Thuật toán đơn định, đa thức).
- Với những bài toán chưa tìm ra thuật toán nhanh, xét xem nó có thuộc lớp NPC?
- Với những bài toán thuộc lớp NPC thì tìm ra thuật toán đa thức để tạo ra lịch biểu gần tối ưu.
- Tìm các ứng dụng của bài toán lập lịch trong thực tế: công nghiệp, giao thông vận tải, hệ điều hành máy tính…

Ví dụ : Người ta đã tìm ra một thuật toán Heuristic giải bài toán trên như sau:

Thuật toán Moore : Tìm lịch biểu trong thời gian đa thức

① Sắp xếp các ô tô theo thứ tự không giảm của các d_i: Thứ tự các ôtô $x_1, x_2, \ldots x_n$ mà $d_1 \leq d_2 \leq \ldots \leq d_n$ ta gọi là dãy hiện thời.

② Trên dãy hiện thời, tìm ô tô đầu tiên quá hạn, giả sử đó là ô tô thứ q. Trong số q ô tô đầu tiên hãy loại ra ô tô nào có thời gian sửa chữa t nhiều nhất (nếu có nhiều ôtô như vậy thì chỉ chọn một).

③ Bước 2 được lặp lại cho đến khi trong dãy hiện thời không còn ô tô nào quá hạn.

④ Các ô tô bị loại ra trước đó được xếp vào sau dãy hiện thời tại cuối bước 3.

Sau bước 4 ta có một thứ tự các ô tô (lịch biểu) cần tìm trong thời gian đa thức (phụ thuộc bước sắp xếp các công việc $O(n^2)$)

<u>Tóm lại</u>: Các đối tượng được xử lí như "ô tô", "chương trình"… được quy về tên chung là "công việc" (job). Các phương tiện xử lí như "nhà máy", "máy tính"…, kể cả

con người có tên chung là "máy" (machine). Khi đó ta có các quan niệm rộng hơn về bài toán lập lịch.

4.2. Phân lớp bài toán lập lịch dạng tĩnh

Giả sử có n công việc J_i (i=1,2,…,n) được xử lí trên m máy M_j (j=1,2,…,m) Giả thiết rằng tại một thời điểm, mỗi công việc được xử lí trên nhiều nhất là 1 máy và mỗi máy có thể xử lí nhiều nhất là một công việc.

Khi đó bài toán lập lịch có thể mô tả dưới dạng mô hình chung : $\alpha|\beta|\gamma$

trong đó α : Biểu diễn mối quan hệ giữa các máy.

β : Biểu diễn mối quan hệ giữa các công việc.

γ : Tiêu chuẩn tối ưu của lịch biểu cần tìm.

4.2.1. Thông tin về công việc

Với mỗi công việc J_i có các tham số sau:

m_i : số máy cần thiết để xử lí công việc J_i .

m_i : thời gian xử lí công việc J_i .

t_{ij} : thời gian xử lí công việc J_i trên máy M_j .

r_i : thời điểm có thể bắt đầu xử lí công việc J_i .

d_i : hạn định mà tại đó J_i phải hoàn thành.

W_i : trọng số của công việc J_i (tiền phạt, tiền thưởng,…).

b_i : thời điểm bắt đầu thực sự xử lí công việc J_i .

c_i : thời điểm hoàn thành công việc J_i .

Giả thiết các tham số đều là biến nguyên (Integer).

4.2.2. Quan hệ giữa các máy

Tham số $\alpha = \alpha_1\alpha_2$ chỉ mối quan hệ giữa các máy.

a) Trường hợp $\alpha_1 \in \{\phi, P, Q, R\}$

+ $\alpha_1 = \phi$: được hiểu là hệ thống chỉ có một máy và $t_{ij} = t_i$.

+ α_1 =P : hệ thống gồm m máy song song đồng nhất $t_{ij} = t_i$, i=1,2,…m.

(Parallel - shop: mỗi công việc có thể xử lí trên một trong số m máy giống nhau).

+ α_1 =Q : hệ thống gồm m máy song song đồng đều $t_{ij} = t_i * q_j$, q_j là tốc độ của máy m_j.

+ α_1 =R : hệ thống gồm m máy song song độc lập (không đồng đều)

b) Trường hợp α_1 ***=0, hệ thống gọi là "Open Shop":***

Trường hợp này không quan tâm đền thứ tự thực hiện các công đoạn của mỗi công việc, trong đó:

- Mỗi công việc J_i đều gồm có m công đoạn: $\{O_{1i}, O_{2i}, \ldots, O_{mi}\}$.
- Mỗi công đoạn O_{ij} được xử lí trên máy M_j trong thời gian t_{ij} .

c) $\alpha_1 \in$ ***{F, J}***

Trường hợp này phải quan tâm đến thứ tự thực hiện các công đoạn của mỗi công việc

+ α_1=F: hệ thống gọi là "Flow shop".

(Hệ dây chuyền: mọi công việc xử lí trên tất cả các máy theo cùng một thứ tự).

Trong đó : - Mỗi công việc J_i gồm tập m_i công đoạn $\{O_{1i}, O_{2i}, \ldots, O_{mi}\}$

- Mỗi O_{ij} được xử lí trên máy M_i trong thời gian t_{ij} .

+ α_1=J, hệ thống được gọi là "Job shop" (mỗi công việc có một tập công đoạn riêng), trong đó:

- Mỗi công việc J_j gồm một tập m_j công đoạn : $\{O_{1j}, O_{2j}, \ldots, O_{mj}\}$
- Mỗi O_{ij} được xử lí trên máy M_{ij} trong thời gian t_{ij} , $M_{i-1,j} \neq M_{ij}$.

d) Trường hợp α_2 ***là số nguyên, dương*** thì m là hằng số và m=α_2 (biết số lượng máy cụ thể)

+ $\alpha_2 \neq \phi$: m là biến thay đổi.

Hiển nhiên α_2=1 $\Leftrightarrow$ $\alpha_1 = \phi$ (vì chỉ có 1 máy nên không chỉ ra mối quan hệ giữa các máy).

4.2.3. Quan hệ giữa các công việc

Tham số $\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ chỉ mối quan hệ giữa các công việc được xác định như sau:

a) $\beta_1 \in$ **{**$P_m t_n$ **,** ϕ**}**

+ β_1=$P_m t_n$: một công việc đang được thực hiện có thể bị ngắt để ưu tiên (Preemption) thực hiện một công việc khác và sau đó nó được tiếp tục xử lí tại các thời điểm tiếp theo cho đến khi hoàn thành.

+ $\beta_1 = \phi$: chỉ sự thực hiện công việc không bị ngắt.

b) $\beta_2 \in$ **{Prec, Tree,** ϕ**}**

+ β_2=Prec: thứ tự ưu tiên thực hiện các công việc được biểu diễn bằng một đồ thị định hướng với tập đỉnh {1,2,…,n}. Nếu G là một đường định hướng từ j $\rightarrow$k thì ta viết

$J_j < J_k$ và đòi hỏi J_j phải được kết thúc trước khi J_k bắt đầu.

+ β_2=Tree: thứ tự ưu tiên thực hiện các công việc được biểu diễn như một cây.

+ $\beta_2 = \phi$: Không có thứ tự ưu tiên.

c) $\beta_3 \in \{r_j \geq 0, \phi\}$**:**

+ $r_j \geq 0$ nghĩa là $r_i \neq r_j$, $\forall i \neq j$: các công việc nhập vào trong hệ thống là không đồng thời.

+ $\beta_3 = \phi$ nghĩa là $r_j = 0, \forall j$: các công việc nhập vào trong hệ thống là đồng thời.

d) $\beta_4 \in \{m_j \leq \overline{m}, \phi\}$

+ $m_j \leq \overline{m}$: chỉ cận trên của m_j (cho trường hợp $\alpha_1 = J$)

+ $\beta_4 = \phi$: không chỉ ra cận trên.

e) $\beta_5 \in \{t_{ij}=1, \underline{t} \leq t_{ij} \leq \overline{t}, \phi\}$

+ t_{ij} =1: chỉ mỗi công đoạn thực hiện hết một đơn vị thời gian.

+ $\underline{t} \leq t_{ij} \leq \overline{t}$: chỉ ra cận dưới và cận trên của t_{ij} .

+ $\beta_5 = \phi$: không chỉ ra 1 cận như vậy.

4.2.4. Một số tiêu chuẩn tối ưu

1. $C_{max} = Max\{C_i\}$ Thời điểm cực đại kết thúc mọi công việc.
2. $\overline{C} = (\frac{1}{n})*\sum_{i=1}^{n} C_i$ Thời điểm trung bình kết thúc mọi công việc.
3. $F_i = C_i - r_i$ Thời gian lưu lại công việc i trong hệ.
4. $\overline{F} = (\frac{1}{n})*\sum_{i=1}^{n} F_i$ Thời gian lưu lại trung bình mọi công việc.
5. $L_i = C_i - d_i$ Mức chậm trễ so với hạn định của công việc i.
6. $L_{max} = Max\{L_i\}$ Mức chậm trễ cực đại.
7. $\overline{L} = (\frac{1}{n})*\sum_{i=1}^{n} L_i$ Mức chậm trễ trung bình.
8. $T_i = Max\{0, L_i\}$ Mức chậm trễ không âm.
9. $T_{max} = Max\{T_i\}$ Mức chậm trễ không âm cực đại.

10. $\sum_{i=1}^{n} w_i u_i$ Lượng tiền phạt (w_i – tiền phạt của công việc i)

trong đó

$$U_i = \begin{cases} 0 & \text{nếu đúng hạn} \\ 1 & \text{nếu quá hạn} \end{cases}$$

4.2.5. Một số ví dụ

1) Hệ 1 | Prec| L_{max}

Cực tiểu hóa mức chậm trễ cực đại của mọi công việc trên một máy. Bài toán này có thể giải trong thời gian đa thức.

2) Hệ R| $P_m t_n$ | $\sum C_i$

Cực tiểu hóa thời điểm kết thúc toàn bộ công việc trên hệ m máy song song độc lập trong trường hợp ngắt.

3) Hệ 3 | t_{ij} =1| C_{max}

Cực tiểu hóa thời điểm cực đại kết thúc mọi công việc trên hệ 3 máy với thời gian xử lí mỗi công đoạn bằng một đơn vị thời gian.

4.2.6. Một số thuật toán lập lịch

- Thuật toán lập lịch tối ưu chậm: Là thuật toán tìm lịch biểu tối ưu, nhưng thời gian xử lí chậm (Độ phức tạp hàm mũ).
- Thuật toán lập lịch tối ưu nhanh: Là thuật toán tìm lịch biểu tối ưu nhưng thời gian xử lí nhanh (Độ phức tạp đa thức).
- Thuật toán lập lịch xấp xỉ nhanh: Là thuật toán tìm lịch biểu gần đúng nhưng thời gian xử lí nhanh (Độ phức tạp đa thức).

Nói chung đa số các bài toán lập lịch thuộc lớp NPC, nên chỉ có một số ít giải được bằng thuật toán tối ưu nhanh, còn lại phải ta phải tìm các thuật toán xấp xỉ nhanh để tìm lịch biểu gần đúng trong thời gian đa thức.

4.3. Một số bài toán lập lịch giải bằng thuật toán lập lịch tối ưu nhanh

4.3.1. Hệ 1,n | | C_{max}

a) <u>Bài toán:</u>

- Input: Cho n công việc 1, 2,…, n xử lý trên một máy, với 3 tham số

  r_i=0, d_i, t_i, $\forall$ i=1, 2,…,n.

- Output: Lập lịch biểu S có C_{max} nhỏ nhất.

b) <u>Thiết kế thuật toán:</u>

Để thời điểm cực đại kết thúc mọi công việc là nhỏ nhất đòi hỏi mọi công việc nhập vào hệ thống phải được thực hiện ngay. Nghĩa là công việc nhập vào phải được thực hiện sớm nhất tại thời điểm t=0. Mỗi hoán vị của tập {1, 2,…, n} công việc cho ta một lịch biểu. Thứ tự thực hiện các công việc là nối tiếp nhau, thỏa mãn điều kiện: Thời điểm kết thúc công việc trước trùng với thời điểm bắt đầu thực hiện công việc tiếp theo. Tóm lại lịch biểu tối ưu phải có tính chất:

- Thời điểm bắt đầu b = 0.
- Liên tục (không cho phép ngắt công việc).
- Liên thông (máy làm việc không ngừng).

Thuât toán: Tạo lịch biểu tối ưu S

Lấy ngay S với tập lập biểu được là {1, 2,…, n}. Các thực hiện được xác định như sau:

Begin

$b_1:=r_1:=0;$

$c_1:=b_1+t_1:=t_1;$

For i:=2 **to** n **do**

Begin

$b_i:=c_{i-1};$

$c_i:=b_i+t_i;$

End;

End;

c) Chứng minh tính đúng đắn:

Phải chỉ ra rằng lịch biểu theo cách xây dựng như trên có 3 tiêu chuẩn thực sự cho C_{max} là nhỏ nhất.

Thật vậy: Lịch biểu S theo cách xây dựng trên có $(C_{max})_S = \sum_{i=1}^{n} t_i$

Giả sử có lịch biểu S' khác lịch biểu S thì $(C_{max})_S<(C_{max})_{S'}$ (hiển nhiên).

d) Độ phức tạp: Phụ thuộc vào kích thước của bài toán ~ O(n).

e) Ví dụ: Xét hệ 1 | 3 với các tham số cho trong bảng:

T/số C/việc	r_i	d_i	t_i
1	0	4	2
2	0	8	5
3	0	12	10

Mỗi hoán vị cho ta một lịch biểu tối ưu $\Rightarrow$ với hệ 3 công việc sẽ có 6 lịch biểu tối ưu.

Lịch biểu $S(x_1,x_2,x_3)$

T/số C/việc	r_i	d_i	t_i	b_i	c_i
x_1	0	4	2	0	2
x_2	0	8	5	2	7
x_3	0	12	10	7	17

$$C_{max}=Max\{C_i\}=17$$

Tương tự: TGB $S=(x_1,x_2,x_3)$ có $C_{max}=17$.

…………

4.3.2. Nhóm hệ 1, n | | L_{max} và 1, n | | T_{max}

<u>a) Biểu diễn bài toán</u>

*** Bài toán 1,n | | L_{max}**

- Input: Cho n công việc 1, 2,…,n với 3 tham số $r_i=0$, d_i, t_i, $\forall$ i=1,2,…,n.
- Output: Lập lịch biểu có L_{max} là nhỏ nhất.

Trong đó: $L_i=C_i-d_i$ là mức chậm trễ so với hạn định của công việc i.

$L_{max}=\underset{1\le i\le n}{Max}\{L_i\}$ là mức chậm trễ cực đại so với hạn định của công việc i

*** Bài toán 1, n | | T_{max}**

- Input: Cho n công việc 1, 2,…, n với 3 tham số $r_i=0$, d_i, t_i, $\forall$ i=1,2,…,n.

- Output: Lập lịch biểu có T_{max} nhỏ nhất.

Trong đó: T_i=Max{0,L_i} là mức chậm trễ không âm của công việc i.

T_{max}=Max{T_i} là mức chậm trễ không âm cực đại.

<u>* Bổ đề:</u> Bài toán 1 ,n││ L_{max} tương đương đa thức với bài toán 1,n││ T_{max}

Chứng minh: Ta biết T_{max}=Max_i{T_i} và T_i=Max_i(0,L_i)

Vậy T_{max}=Max_i{Max(0, L_i)}= max{Max_i(0, L_i)}= max(0, L_{max})

Do vậy nếu tạo được lịch biểu có T_{max} là nhỏ nhất thì L_{max} cũng nhỏ nhất và ngược lại

$\Rightarrow$ Bài toán 1 ,n││ L_{max} và 1,n││ T_{max} là tương đương đa thức. Vì vậy ta chỉ cần xét một bài toán.

b) Bài toán 1,n││ L_{max}

<u>* Thiết kế thuật toán: </u> Lịch biểu S là tối ưu nếu nó liên tục, điểm bắt đầu b=0 và liên thông, đồng thời các công việc được sắp xếp theo thứ tự không giảm hạn định của chúng, nghĩa là $d_1 \le d_2 \le ... \le d_n$.

```
Begin
   1. Sắp xếp các công việc theo thứ tự không giảm các hạn định của chúng
   2. Tạo các thực hiện
      b1:=0; c1:=b1+t1;
      L1:=c1- d1;
               For i:=2 to n do
                  Begin
                           bi:=ci-1; ci:=bi+ti;
                           Li:=ci-di;
                  End;
      For i:=1 to n do Lmax:=Max{Li};
 End;
```

<u>Chứng minh tính đúng đắn:</u> Chứng minh rằng lịch biểu theo cách xây dựng trên sẽ cho L_{max} nhỏ nhất.

Ta sẽ sử dụng một kỹ thuật chứng minh cơ bản trong lý thuyết lập lịch: *Thủ thuật hoán vị* để chứng minh tính đúng đắn của thuật toán trên

- Giả sử S là lịch biểu mà các công việc không đươc sắp xếp theo thứ tự tăng dần của hạn định. Nghĩa là $\exists$k sao cho $d_k > d_{k+1}$ (*). Ta sẽ chứng minh có lịch biểu S' tốt hơn S .

- Giả sử x_k và x_{k+1} là hai công việc ứng với 2 vị trí k và k+1 trong lịch biểu S.
- Xét lịch biểu S' khác S ở chỗ: Trong S' các công việc x_k và x_{k+1} được thực hiện tương ứng ở các vị trí k+1 và k.
- Trong cả hai lịch biểu S và S': Thứ tự thực hiện (k-1) công việc đầu và n-(k+1) công việc cuối là như nhau.
- Trong cả hai lịch biểu S và S', điểm đầu và điểm cuối của n – 2 công việc là như nhau $\Rightarrow$ Mức chậm trễ của các công việc cũng như nhau. Ta gọi mức chậm trễ lớn nhất của n-2 côngviệc đó là L.

Cần chứng minh: $Max\{L, L_k, L_{k+1}\}_{S'} \leq Max\{L, L_k, L_{k+1}\}_S$ trong đó L_k và L_{k+1} là mức chậm trễ của công việc x_k và x_{k+1}.

Thật vậy: Đặt $T=\sum_{i=1}^{k-1} t_i$, khi đó ta có:

$L_k(S)=T+t_k-d_k$ (1). S : ______ d_k,>d_{k+1} ______

$L_{k+1}(S)=T+t_k+t_{k+1}- d_{k+1}$ (2). S': ______ d_{k+1},<d_k ______

$L_k(S') = T+t_k+t_{k+1}-d_k$ (3).

$L_{k+1}(S') =T+t_{k+1}- d_{k+1}$ (4).

Từ (2) và (3) ta có: $L_k(S')<L_{k+1}(S)$ vì $d_k>d_{k+1}$ (5).

Từ (2) và (4) ta có: $L_{k+1}(S')<L_{k+1}(S)$ (6).

Từ (5) và (6) $\Rightarrow L_{k+1}(S) > Max\{L_k(S'),L_{k+1}(S')\}$

$\Rightarrow Max\{L, L_k, L_{k+1}\}_{S'} \leq Max\{L, L_k, L_{k+1}\}_S$

Qua đó ta thấy S' tôt hơn S. Nếu xét lịch biểu S' mà một cặp công việc nào đó thỏa mãn (*) thì ta được S" tốt hơn S'….

Sau hữu hạn bước ta được S* là tốt nhất và S* là lịch biểu tối ưu cần tìm.

<u>Độ phức tạp:</u> Phụ thuộc vào bước sắp xếp các công việc : $O(n^2)$

<u>Ví dụ</u>: Xét hệ $1 | 3$ với các tham số cho trong bảng sau:

T/số \ C/ việc	r_i	t_i	d_i
x_1	0	3	4
x_2	0	2	8
x_3	0	1	3

- Sắp xếp công việc theo d_i: x_3, x_1, x_2 có L_{max}= 0 là TGB tối ưu cần tìm.

4.3.3. Hệ $1,n \mid r_i \geq 0 \mid C_{max}$

Hệ có n công việc được xử lí trên 1 máy, các công việc nhập vào trong hệ là không đồng thời ($r_i \geq 0$, $\forall$ i=1, 2,…,n). Lập lịch biểu để thời điểm cực đại kết thúc mọi công việc là nhỏ nhất

Input: n công việc 1, 2,…, n, với các tham số $r_i \geq 0$, d_i, t_i, $\forall$ i=1, 2,…,n.

Output: Lập lịch biểu có C_{max} nhỏ nhất, $C_{max}=Max_i\{C_i\}$.

a) Thiết kế thuật toán

Để thời điểm cực đại kết thúc mọi công việc là nhỏ nhất ta sắp xếp các công việc theo thứ tự không giảm của các r_i: $r_1 \leq r_2 \leq ... \leq r_n$. Như vậy, xây dựng lịch biểu tối ưu là sắp xếp các công việc theo thứ tự không giảm thời điểm có thể bắt đầu thực hiện chúng.

Begin

1. Sắp xếp các công việc không giảm theo thời điểm có thể bắt đầu các công việc : $r_1 \leq r_2 \leq ... \leq r_n$

b_1:=r_1; c_1:=b_1+t_1;

For i:=2 **to** n **do**

Begin

b_i:=Max(c_{i-1}, r_i);

c_i:=b_i+t_i;

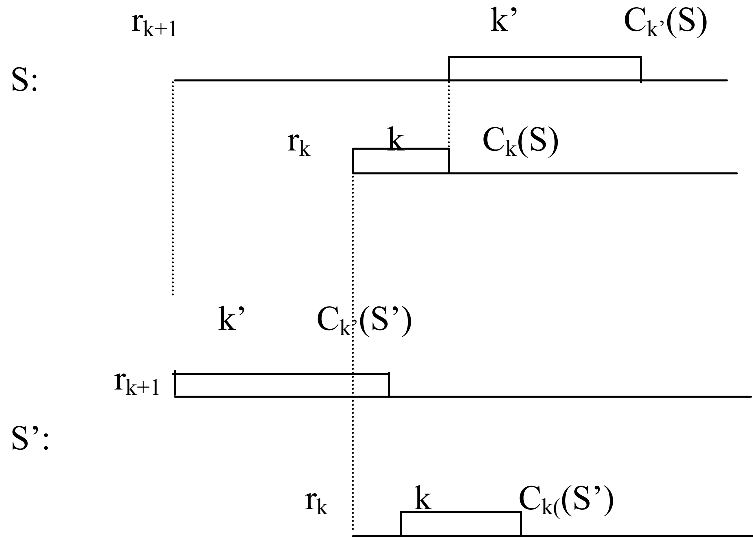
End;

End;

b) Chứng minh tính đúng đắn:

Ta chứng minh rằng: Nếu sắp xếp các công việc theo thứ tự không giảm của các r_i thì C_{max} là nhỏ nhất.

Thật vậy: Gọi S là lịch biểu không theo thứ tự trên. Tức là $\exists k: r_{k+1} < r_k$ (*)



Giả sử S' khác S ở chỗ k và k' được thực hiện ở vị trí k+1 và k tương ứng. Trong cả 2 lịch biểu thứ tự thực hiện (k-1) công việc đầu và n-(k+1) công việc cuối là như nhau. Trong cả 2 lịch biểu điểm đầu và điểm cuối của n-2 công việc là như nhau: Ta phải chứng minh $(C_{max})_{S'} \leq (C_{max})_S$

Thật vậy:

Trong S: $C_k(S) = Max(C_{k-1}, r_k)+t_k$

$C_{k'}(S) = Max(C_k(S), r_{k+1})+t_{k'}$

Trong S': $C_{k'}(S')=Max(C_{k-1}, r_{k+1})+t_{k'}$

$C_k(S')=Max(C_{k'}(S'), r_k)+t_k$

Nếu $C_{k-1}<r_k$ thì: $C_k(S)=r_k+t_k$

$C_{k'}(S)=r_k+t_k+t_{k+1}$

$C_{k'}(S')=Max(C_{k-1}, r_{k+1})+t_{k+1}<r_k+t_{k+1}$

$C_k(S')<r_k+t_{k+1}+t_k$

$\Rightarrow C_k(S') \leq C_{k'}(S) \Rightarrow (C_{max})_{S'} \leq (C_{max})_S$

Nếu $C_{k-1} \geq r_k \Rightarrow (C_{max})_{S'}=(C_{max})_S$

$\Rightarrow$ Rõ ràng S' tốt hơn S.

Nếu trong S' xét cặp công việc nào đó thỏa mãn (*) và chứng minh tương tự ta được S" tốt hơn S', cứ tiếp tục như vậy thì sau hữu hạn bước ta sẽ được S* tốt hơn cả và S* là lịch biểu tối ưu cần tìm.

<u>c) Độ phức tạp:</u> Phụ thuộc vào bước sắp xếp các công việc : $O(n^2)$

d) Ví dụ:

T/gian \ C/ việc	r_i	d_i	t_i
1	0	5	3
2	2	6	1
3	6	3	3
4	3	7	5
5	0	5	1

T/gian \ C/ việc	r_i	d_i	t_i	c_i
1	0	5	3	3
5	0	5	1	4
2	2	6	1	5
4	3	7	5	10
3	6	3	3	13

$\Rightarrow$ Lịch biểu S (1, 5, 2, 4, 3) có C_{max}=13 nhỏ nhất.

4.4. Bài toán lập lịch gia công trên 2 máy, thuật toán Johnson

4.4.1. Bài toán 2; F$|r_i \geq 0|$ C_{max}

Input:

- Cho n chi tiết $D_1, D_2, \ldots, D_n$ lần lượt gia công trên 2 máy A, B.
- Thời gian gia công chi tiết D_i trên A là t_{ai} , trên B là t_{bi}
- Trình tự gia công các chi tiết trên 2 máy là như nhau.

Output: Tìm lịch biểu(trình tự gia công) các chi tiết trên 2 máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể được (C_{max} đạt Min)

4.4.2. Thiết kế thuật toán

a) Phân tích bài toán

- Mỗi lịch gia công tương ứng với một hoán vị của các chi tiết $\pi = (\pi(1), \pi(2), ..., \pi(n))$

- Kí hiệu b_{jX} : Thời điểm bắt đầu chi tiết j trên máy X

C_{jX} : Thời điểm kết thúc chi tiết j trên máy X.

(X=A, B; j =1, 2,…, n)

- Máy A bắt đầu thực hiện công việc tại thời điểm $b_{\pi A} = r_{\pi A} = 0$. Các công việc được thực hiện nối tiếp nhau: $C_{\pi(j-1)A} \le b_{\pi(j)A}$, j = 2,3,…, n. (1).
- Máy B có thể bắt đầu thực hiện công việc $\pi(1)$ ngay sau khi máy A kết thúc công việc gia công $\pi(1)$ vào thời điểm: $b_{\pi(1)B} \ge C_{\pi(1)A}$ (2).

và bắt đầu gia công chi tiết $\pi(k)$ (k=1, 2,…, n) sau khi công việc này được thực hiện xong trên máy A và đồng thời phải hoàn thành gia công chi tiết $\pi(k-1)$, tức là: $b_{\pi(k)B} \ge Max(C_{\pi(k)A}, t_{\pi(k-1)B})$, k=2,3,…, n. (3).

- Thời gian hoàn thành việc gia công tất cả các chi tiêt trên 2 máy là $C(\pi) = C_{\pi(n)B}$.

Rõ ràng với π cố định, $C(\pi)$ đạt min khi tất cả các dấu bất đẳng thức ở (1), (2), (3) được thay bằng dấu đẳng thức:

$$
\begin{aligned}
&b_{\pi(1)A} = 0 \\
&b_{\pi(k)A} = C_{\pi(k-1)A}, k = 2,3,...,n \\
&b_{\pi(1)B} = C_{\pi(1)A} \\
&b_{\pi(k)B} = Max(C_{\pi(k)A}, C_{\pi(k-1)B}), k = 2,3,...,n
\end{aligned}
$$

Các máy sẽ thực hiện ngay các công việc một khi điều kiện cho phép.

b) Thiết kế thuật toán

- Nếu thực hiện việc gia công các chi tiết theo lịch hoán vị các công việc trên máy A, sau đó thực hiện với trình tự tương ứng trên máy B thì trên máy B sẽ có nhiều thời gian chết.
- Nếu để 2 máy hoạt động liên tục thì phải dồn thời gian chết trên máy B vào đoạn đầu (điều này không làm tăng thời gian hoàn thành lịch gia công). Máy A bắt đầu vào thời điểm b_A=0

$$\Rightarrow C(\pi) = b_{B(\pi)} + \sum_{j=1}^{n} t_{Bj}$$

Dễ thấy $b_{B(\pi)} = C_{\pi(1)A} + \sum$ thời gian chết trên máy B

$\Rightarrow$ Tìm Min{ $b_{B(\pi)}$ }?

**** Bổ đề 1:*** Giả sử $\pi = (\pi(1),....,\pi(k-1),\pi(k),\pi(k+1),....,\pi(n))$ là một lịch gia công còn π' là lịch gia công thu được từ π bằng cách hoán vị 2 phần tử $\pi(k)$ và $\pi(k+1)$: $\pi' = (\pi(1),....,\pi(k-1),\pi(k+1),\pi(k),....,\pi(n))$.

Khi đó nếu $Min(a_{\pi(k)}, b_{\pi(k+1)}) \le Min(b_{\pi(k)}, a_{\pi(k+1)})$ thì $b_{B(\pi)} \le b_{B(\pi')}$

***Bổ đề 2**:* Nếu i, j , k là 3 chỉ số thỏa mãn

$\min(a_i, b_j) \le \min(a_j, b_i)$

$\min(a_j, b_k) \le \min(a_k, b_j)$

thì $\min(a_i, b_k) \le \min(a_k, b_i)$

Định lý Johnson (1954):

C(π) đạt giá trị nhỏ nhất khi gia công $\pi = (\pi(1), \pi(2), ..., \pi(n))$ thỏa mãn $\min(a_{\pi(k)}, b_{\pi(k+1)}) \le \min(b_{\pi(k)}, a_{\pi(k+1)})$ với mọi k=1, 2,…, n-1.

Từ đó ta có cơ sở để xây dựng thuật toán như sau:

Giả sử $x = \min_{1 \le i \le n}(a_i, b_i)$. Xét 2 trường hợp:

1. Nếu x=a_k, k tùy ý $\Rightarrow$ $\min(a_k, b_j) \le \min(b_k, a_j)$, $\forall j \ne k$ $\Rightarrow$Chi tiết D_k phải được gia công đầu tiên trong lịch biểu tối ưu.
2. Nếu $x = b_p$ với p nào đó $\Rightarrow$ $\min(a_p, b_j) \ge \min(b_p, a_j)$ $\forall j \ne p \Rightarrow$ Chi tiết D_p phải được gia công cuối cùng.

Thuật toán Johnson:

①. Chia các chi tiết thành 2 nhóm:

- Nhóm N_1 gồm các chi tiết D_i thỏa mãn a_i<b_i tức là $\min(a_i, b_i)=a_i$.
- Nhóm N_2 gồm các chi tiết D_i thỏa mãn a_i>b_i tức là $\min(a_i, b_i)=b_i$.

Các chi tiết D_i thỏa mãn $a_i=b_i$ xếp vào nhóm nào cũng được.

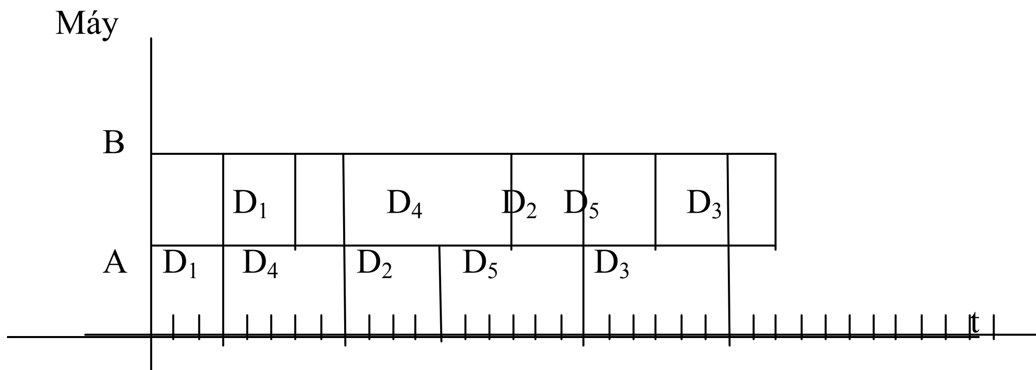
②. Sắp xếp các chi tiết trong N_1 theo chiều tăng của a_i và sắp xếp các chi tiết trong N_2 theo chiều giảm b_i.

③. Nối N_2 vào đuôi N_1, dãy thu được (trái→phải) là lịch gia công tối ưu.

Ví dụ: Cho 5 chi tiết phải gia công trên 2 máy . Thời gian gia công các chi tiết trên các máy được cho trong bảng sau:

Chi tiết / Mỏy	D_1	D_2	D_3	D_4	D_5
A	3	4	6	5	6
B	3	3	2	7	3

Sơ đồ Gantt của lịch biểu sau khi thực hiện thuật toán Johnson như sau:



Kết quả: Thời gian hoàn thành việc gia công là $C_{max} = 26$.

Chú ý:

- Có thể có nhiều lịch biểu tối ưu, chúng có thể khác nhau về thời điểm bắt đầu của máy B nhưng đều chung nhau một thời điểm kết thúc (lịch biểu π' =(D_4, D_1, D_5, D_2, D_3)).
- Có thể chứng minh được rằng việc tìm lịch gia công dưới dạng mỗi máy một trình tự gia công riêng không dẫn tới việc hoàn thành gia công các chi tiết sớm hơn. Vì vậy thuật toán Johnson vẫn cho kết quả đúng của bài toán mà không cần có giả thiết rằng trình tự gia công trên hai máy phải như nhau.
- Không thể thu được định lý tương tự như định lý Johnson cho trường hợp bài toán 3 máy hoặc nhiều hơn. Trong trường hợp tổng quát, hiện nay chưa có phương pháp hữu hiệu nào để giải chúng ngoài việc sử dụng phương pháp "vét cạn" hoặc nhánh cận.

4.4.3. Một số trường hợp riêng có thể dẫn về bài toán 2 máy

Xét bài toán gia công n chi tiết trên 3 máy theo thứ tự A, B, C với bảng thời gian a_i, b_i, c_i , i= 1, 2,…, n thoả mãn: $\max b_i \le \min a_i$ hoặc $\max b_i \le \min c_i$

Tức là thời gian gia công của máy B khá nhỏ so với A hoặc C.

Khi đó, lịch gia công trên 3 máy sẽ trùng với lịch gia công tối ưu trên hai máy: máy thứ nhất với thời gian $a_i + b_i$ và máy thứ hai với thời gian $b_i + c_i$ (lưu ý chỉ có lịch tối ưu của chúng là trùng nhau còn thời gian gia công của chúng là khác nhau).

Ví dụ: Thời gian gia công 5 chi tiết trên các máy A, B, C cho bởi bảng sau:

Chi tiết / Máy	D_1	D_2	D_3	D_4	D_5
A	7	11	8	7	6
B	6	5	3	5	3
C	4	12	7	8	3

Ta thấy: max $b_i = 6 \leq$ min $a_i = 6$, do đó bài toán được dẫn về việc tìm lịch gia công tối ưu trên hai máy A', B'

Chi tiết / Máy	D_1	D_2	D_3	D_4	D_5
A'	13	16	11	12	9
B'	10	17	10	13	6

Lịch gia công tối ưu tìm được là $\pi = (D_4 , D_2 , D_1 , D_3 , D_5)$ với thời gian hoàn thành $C_{max} = 49$.