

Phân tích và thiết kế giải thuật

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường Đại học Bách khoa
Đại học Đà Nẵng

Mục đích

- ▣ Các khái niệm liên quan đến bài toán và giải quyết bài toán
- ▣ Phân tích và đánh giá giải thuật
- ▣ Các kỹ thuật thiết kế giải thuật
- ▣ Vận dụng giải quyết các bài toán cụ thể

Nội dung

- ▣ Giới thiệu
- ▣ Độ phức tạp (complexity)
- ▣ đệ quy (recursion)
- ▣ Chia để trị (divide and conquer)
- ▣ Quy hoạch động (dynamic programming)
- ▣ Thuật toán tham lam (greedy algorithms)
- ▣ Quay lui (backtracking)

3

Tài liệu tham khảo

1. **Introduction to algorithms, T.H. Cormen, C.E. Leiserson, R.R. Rivest, Mit Press 1990.**
2. Type de Données et Algorithmes, M-C Gaudel, M. Soria, C. Froidevaux, Ediscienne international, 1993.
3. Cours Complexité, M. Daniel, ESIL, 2006.
4. Data structures and algorithms, Alfred V. Aho, John E. Hopcroft, Addison-Wesley, 1983.
5. Algorithm Analysis and Computational Complexity, Ian Parberry, Lecture Notes, University of North Texas, 2001.
6. The Art of Computer Programming, Volume 2, D. Knuth, Addison-Wesley, 1981.
7. Các tài liệu khác trên Internet.

4

Giới thiệu (1)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Giới thiệu

- Khái niệm giải thuật/thuật toán (algorithm)
 - Thuật toán là một dãy xác định các thao tác cơ bản áp dụng trên dữ liệu vào nhằm đạt được giải pháp cho một vấn đề
 - Hai vấn đề
 - Tìm một phương pháp giải quyết vấn đề
 - Giải pháp cho $ax^2 + bx + c = 0$: rõ ràng và xác định
 - Giải pháp cho $ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$: không có giải pháp tổng quát
 - Tìm một giải pháp **hiệu quả**
 - Phân biệt giải thuật và chương trình
 - Chương trình là cài đặt thuật toán bằng một ngôn ngữ lập trình

6

Giới thiệu

□ Thuật toán

- Thủ tục tính toán nhận tập các **dữ liệu vào** (input) và tạo các **dữ liệu ra** (output)
- Thuật toán được gọi là **đúng đắn** (correct), nếu **thuật toán dừng** cho **kết quả đúng** với mọi dữ liệu vào

7

Giới thiệu

□ Thuật toán hằng ngày trong cuộc sống

- Nấu cơm
 - ...
- Gọi điện thoại

□ Thuật toán trong toán học/tin học

- Nhân hai ma trận
- Tính tích phân
- Giải hệ phương trình bậc nhất
- ...

8

Giới thiệu

▣ Các tính chất của thuật toán

- Tính tổng quát
- Tính hữu hạn
- Tính không nhập nhằng
- Tính hiệu quả

9

Giới thiệu

▣ Tính tổng quát (1)

- Thuật toán phải áp dụng cho tất cả các mẫu dữ liệu vào chứ không chỉ là một mẫu dữ liệu vào cụ thể
- Ví dụ
 - ▣ Sắp xếp tăng dần một dãy các giá trị
 - ▣ Dữ liệu vào : 2 1 4 3 5
 - ▣ Kết quả : 1 2 3 4 5

10

Giới thiệu

□ Tính tổng quát (2)

■ Phương pháp

- So sánh hai phần tử liên tiếp nhau từ trái qua phải, nếu không đúng thứ tự thì hoán đổi vị trí chúng

Bước 1: 2 ↔ 1 4 3 5
 Bước 2: 1 2 4 3 5
 Bước 3: 1 2 4 ↔ 3 5
 Bước 4: 1 2 3 4 5

- Dãy đã được sắp xếp
- Thuật toán có tổng quát không ?
- Không
 - Ví dụ dữ liệu vào: 2 1 5 4 3

11

Giới thiệu

□ Tính hữu hạn

- Thuật toán phải dừng sau một số bước xác định

■ Ví dụ

```
nhập n
while (n ≠ 0)
  n = n - 2
endwhile
```

- Thuật toán có dừng không ?

12

Giới thiệu

□ Tính không nhập nhằng

- Các thao tác trong thuật toán phải được đặc tả chặt chẽ
 - Ở mỗi bước thực thi, bước tiếp theo sẽ được thực thi phải được xác định rõ ràng

■ Ví dụ

```
Bước 1:  $x = 0$ 
Bước 2: tăng  $x$  lên 1 hoặc giảm  $x$  đi 1
Bước 3: if ( $x \in [-2, 2]$ ) then
           goto Bước 2
```

- Thuật toán có nhập nhằng ?

13

Giới thiệu

□ Tính hiệu quả

- Thuật toán phải sử dụng hiệu quả nguồn tài nguyên máy tính
 - Thời gian
 - Bộ nhớ

■ Ví dụ

```
gt1 (n)
begin
  p = 1
  for i from 1 to n do
    p = p * i;
  endfor
end
```

```
gt2 (n)
begin
  if (n = 0) return (1)
  else
    return (n*gt2(n-1))
  endif
end
```

- Thuật toán nào hiệu quả hơn ?

14

Giới thiệu

□ Đặc tả thuật toán (1)

■ Có nhiều cách đặc tả thuật toán

- Không hình thức
 - Ngôn ngữ tự nhiên
- Nửa hình thức
 - Kết hợp ngôn ngữ tự nhiên và các kí hiệu toán học
 - Sơ đồ khối, ngôn ngữ giả, ...
- Hình thức
 - Các kí hiệu toán học
 - Ngôn ngữ Z, ngôn ngữ B, ...

15

Giới thiệu

□ Đặc tả thuật toán (2)

■ Chọn phương pháp đặc tả nửa hình thức

- Ngôn ngữ giả tựa C/Pascal kết hợp ngôn ngữ tự nhiên

```
if (điều kiện) then
...
else
...
endif
```

```
while () do
...
endwhile
```

```
for ... from ... to ... do
...
endfor
```

16

Giới thiệu

□ Ví dụ 1

- Có 15 hộp đinh có chiều dài khác nhau. Cần đặt chúng vào trong 15 ngăn kéo. Chúng ta cần phải thực hiện thế nào để có thể lấy được loại đinh chúng ta cần một cách nhanh nhất?

17

Giới thiệu

□ Ví dụ 1

- Giải pháp thô
 - Chúng ta xếp các hộp đinh vào các ngăn kéo theo thứ tự bất kỳ. Sau đó, mỗi khi cần lấy đinh chúng ta mở các ngăn kéo theo thứ tự từ trái qua phải.
 - Trường hợp xấu nhất: phải mở 15 ngăn kéo
 - Trường hợp trung bình: giả sử xác suất lấy các loại đinh ngang nhau, phải mở 8 ngăn kéo

$$\frac{1}{15} \sum_{i=1}^{15} i = \frac{1}{15} * \frac{15 * 16}{2} = 8$$

- Nếu những loại đinh trong các ngăn kéo cuối cùng luôn được sử dụng, cần nhiều lần mở các ngăn kéo

18

Giới thiệu

□ Ví dụ 1

■ Giải pháp Las Vegas

- Chúng ta chọn ngăn kéo để mở một cách ngẫu nhiên
 - Nếu may mắn, chúng ta có loại đỉnh cần lấy
 - Nếu không, chúng ta loại bỏ ngăn kéo vừa mở, thực hiện mở ngăn kéo khác một cách ngẫu nhiên

$$\frac{1}{15} * 1 + \frac{14}{15} * \frac{1}{14} * 2 + \frac{14}{15} * \frac{13}{14} * \frac{1}{13} * 3 + \dots = \frac{1}{15} \sum_{i=1}^{15} i = 8$$

19

Giới thiệu

□ Ví dụ 1

■ Giải pháp tiên xử lý

- Chúng ta sắp xếp các hộp đỉnh theo thứ tự chiều dài của chúng
- Xếp vào các ngăn kéo theo thứ tự trên
- Nếu lại mở các ngăn kéo từ trái sang phải thì sẽ không thu được lợi ích gì
- Mở các ngăn kéo theo phương pháp tìm kiếm nhị phân
- Trường hợp xấu nhất: 4 lần mở ngăn kéo
- Trường hợp trung bình: 3.26

$$\frac{1}{15} * 1 + \frac{2}{15} * 2 + \frac{4}{15} * 3 + \frac{8}{15} * 4 = 3.26$$

- Giải pháp chỉ có ý nghĩa khi chúng ta thường xuyên mở các ngăn kéo lấy đỉnh

20

Giới thiệu

□ Ví dụ 2

- Thuật toán nhân hai số nguyên
- Thuật toán nhân Bắc Mỹ

$$\begin{array}{r}
 567 \\
 1234 \\
 \hline
 2268 \\
 1701 \\
 1134 \\
 567 \\
 \hline
 699678
 \end{array}$$

21

Giới thiệu

□ Ví dụ 2

- Thuật toán nhân Anh

$$\begin{array}{r}
 567 \\
 1234 \\
 \hline
 567 \\
 1134 \\
 1701 \\
 2268 \\
 \hline
 699678
 \end{array}$$

22

Giới thiệu

□ Ví dụ 2

■ Thuật toán nhân Ả-rập

		5	6	7			
0	0	5	0	6	0	7	1
6	1	0	1	2	1	4	2
9	1	5	1	8	2	1	3
9	2	0	2	4	2	8	4
		6	7	8			

■ So sánh độ hiệu quả các thuật toán ?

23

Giới thiệu

□ Ví dụ 2

■ Đếm số phép toán nhân và cộng được thực hiện

- Thuật toán nhân Bắc Mỹ và thuật toán nhân Anh đều sử dụng 12 phép nhân và 11 phép cộng
- Thuật toán Ả-rập sử dụng 12 phép nhân và 21 phép cộng
- Số phép toán phụ thuộc vào số chữ số của mỗi số nguyên
 - Số phép nhân bằng $m \times n$ với m và n lần lượt là số chữ số của mỗi số nguyên

24

Giới thiệu

□ Ví dụ 2

■ Thuật toán nhân Nga

567	1234
283	2468
141	4936
70	9872
35	19744
17	39488
8	78976
4	157952
2	315904
1	631808
	699678

Ưu điểm: không cần ghi nhớ các kết quả nhân trung gian, chỉ cần thực hiện phép cộng và chia cho 2

25

Giới thiệu

□ Ví dụ 2

■ Thuật toán nhân « chia để trị »

- Số chữ số của hai số nguyên phải bằng nhau và phải bằng lũy thừa 2
 - Nếu số chữ số của hai số nguyên khác nhau thì thêm vào các số 0 ở bên trái
- Ý tưởng: chúng ta thay phép nhân của hai số nguyên có n chữ số bởi 4 phép nhân của hai số nguyên có $n/2$ chữ số. Tiếp tục, thay phép nhân của hai số nguyên có $n/2$ chữ số bởi 4 phép nhân của hai số nguyên có $n/4$ chữ số, ...
- Cần nhân hai số nguyên: a và b . Gọi a_L và a_R là hai nửa trái và phải của số a , tương tự b_L và b_R là hai nửa trái và phải của b

26

Giới thiệu

□ Ví dụ 2

- Thuật toán nhân « chia để trị »

$$\begin{array}{r}
 \begin{array}{cc} a_L & a_R \\ * & b_L & b_R \end{array} \\
 \hline
 \begin{array}{cc} a_L b_R & a_R b_R \\ + & a_L b_L & a_R b_L \end{array} \\
 \hline
 a_L b_L + (a_L b_R + a_R b_L) + a_R b_R
 \end{array}$$

$$\begin{aligned}
 a * b &= (a_L * 10^{n/2} + a_R) * (b_L * 10^{n/2} + b_R) \\
 &= a_L * b_L * 10^n + a_L * b_R * 10^{n/2} + a_R * b_L * 10^{n/2} + a_R * b_R \\
 &= a_L * b_L * 10^n + (a_L * b_R + a_R * b_L) 10^{n/2} + a_R * b_R
 \end{aligned}$$

Để nhân a và b bởi các nửa của a và b, cần sử dụng 4 phép nhân

27

Giới thiệu

□ Ví dụ 2

- Thuật toán nhân « chia để trị »

- Giảm số phép nhân a và b bởi các nửa của a và b từ 4 xuống còn 3

$$\begin{aligned}
 \text{Đặt: } p &= a_L * b_L \\
 q &= a_R * b_R \\
 r &= (a_L + a_R) * (b_L + b_R)
 \end{aligned}$$

Khi đó:

$$\begin{aligned}
 a * b &= a_L * b_L * 10^n + (a_L * b_R + a_R * b_L) * 10^{n/2} + a_R * b_R \\
 &= p * 10^n + (r - p - q) * 10^{n/2} + q
 \end{aligned}$$

- Số phép nhân sử dụng bởi thuật toán này được đánh giá là $n * m^{0.59}$ với $n \geq m$
- Khi nhân hai số nguyên khá lớn, thuật toán này được đánh giá nhanh hơn các thuật toán trước

28

Giới thiệu

▣ Ví dụ 2

- Thuật toán nhân « chia để trị »

0567 * 1234

nhân	số bước dịch	kết quả
05 12	4	60
05 34	2	170
67 12	2	804
67 34	0	2278
		<hr/>
		699678

05 * 34

nhân	số bước dịch	kết quả
0 3	2	0
0 4	1	0
5 3	1	15
5 4	0	20
		<hr/>
		170

29

Giới thiệu

▣ Ví dụ 3: tính x^n

- Vào: một số nguyên x và một số nguyên không âm n
- Ra: $y = x^n$
- Thuật toán đơn giản

```
y = 1
for i from 1 to n do
    y = y * x
endfor
```

30

Giới thiệu

□ Ví dụ 3: tính x^n

- Thuật toán nhị phân
 - Giảm số phép toán
 - Ý tưởng: sử dụng các kết quả trung gian
 - Nguyên tắc: nếu $n/2 = 0$ thì $x^n = x^{n/2} * x^{n/2}$ nếu không thì $x^n = x^{n-1} * x$
 - Chặng hạn
 - $x^{35} = x^{34} * x$
 - $x^{34} = x^{17} * x^{17}$
 - $x^{17} = x^{16} * x$
 - $x^{16} = x^8 * x^8$
 - $x^8 = x^4 * x^4$
 - $x^4 = x^2 * x^2$
 - $x^2 = x * x$

31

Giới thiệu

□ Ví dụ 3: tính x^n

- Thuật toán nhị phân
 1. Biểu diễn n bởi dãy nhị phân
 2. Thay mỗi bit nhị phân
 - « 1 » bởi các kí hiệu SX
 - « 0 » bởi các kí hiệu S
 3. Xoá cặp kí hiệu SX bên trái nhất
 4. Kết quả: cách tính x^n trong đó
 - S nghĩa là bình phương
 - X nghĩa là nhân với x
 - Bắt đầu từ x

32

Giới thiệu

□ Ví dụ 3: tính x^n

■ Thuật toán nhị phân

Biểu diễn n bởi dãy nhị phân: $e_{k-1}e_{k-2}\dots e_0$

if $e_{k-1} = 0$ **then**
 $y = 1$ // khi $n = 0$

else
 $y = x$

endif

for i **from** $k-2$ **to** 0 **do**

$y = y * y$

if $e_i = 1$ **then**
 $y = y * x$

endif

endfor

33

Giới thiệu

□ Ví dụ 3: tính x^n

■ Thuật toán nhị phân

□ Ví dụ: $n = 35 = 100011_{(2)}$

e_i	y	số phép nhân
1	x	0
0	x^2	1
0	x^4	2
0	x^8	3
1	x^{17}	5
1	x^{35}	7

34

Giới thiệu

□ Ví dụ 3: tính x^n

■ Thuật toán nhị phân: giải thích

- Biểu diễn nhị phân của n : $n = \sum_{i=0}^p A_i 2^i$
- Giả sử chúng ta đang trong quá trình tính x^n . Gọi j là vị trí bit cuối cùng (trái nhất) biểu diễn n , y_j là kết quả cuối cùng có được. Ban đầu: $j = p$, $y_p = x = x^{A_p}$
- Có hai khả năng của A_{j-1}
 - $A_{j-1} = 1$. Thay thế A_{j-1} bởi SX. Vậy $y_{j-1} = y_j^2 * x$
 - $A_{j-1} = 0$. Thay thế A_{j-1} bởi S. Vậy $y_{j-1} = y_j^2$
- Cả hai trường hợp: $y_{j-1} = y_j^2 * x^{A_{j-1}}$
- Vậy: $y_{p-1} = y_p^2 * x^{A_{p-1}} = (x^{A_p})^2 * x^{A_{p-1}} = x^{2A_p + A_{p-1}}$
- Bằng truy hồi:

$$y_1 = x^{\sum_{i=0}^p A_i 2^i} = x^n$$

35

Giới thiệu

□ Ví dụ 3: tính x^n

■ Thuật toán nhị phân: có giải pháp tốt hơn ?

- Chẳng hạn tính x^{15}
 1. $n = 15 = 1111$
 2. SX SX SX SX
 3. SX SX SX
 4. Bắt đầu bởi x , chúng ta có: $x^2, x^3, x^6, x^7, x^{14}, x^{15}$

Vậy, chúng ta sử dụng 6 phép nhân

Tuy nhiên, chúng ta có thể tính x^{15} bởi:

$$x^2, x^3, x^6, x^{12}, x^{15} = x^{12} * x^3$$

Nghĩa là chỉ cần sử dụng 5 phép nhân

36

Giới thiệu

□ Ví dụ 3: tính x^n

■ Phương pháp ước số

- $x^n = x$ nếu $n = 1$
- $x^n = x^{n-1} * x$ nếu n là số nguyên tố
- $x^n = (x^r)^s$ nếu $n = r * s$ với r là ước số nguyên tố nhỏ nhất của n và s lớn hơn 1

■ Ví dụ: $n = 15$

- $15 = 3 * 5$, khi đó $x^{15} = (x^3)^5$. Áp dụng thuật toán để tính x^3 và c^5 với $c = x^3$.
- $x^3 = x^2 * x$, áp dụng thuật toán tính x^2
- $x^2 = x * x$
- $c^5 = c^4 * c$, áp dụng thuật toán tính c^4
- $c^4 = (c^2)^2$
- 2 phép nhân tính c^4 , 3 phép nhân tính c^5 , 2 phép nhân tính x^3 . Vậy 5 phép nhân tính x^{15}

37

Giới thiệu

□ Phân tích và đánh giá các thuật toán

- Dựa vào các tính chất của thuật toán
- Chứng minh sự đúng đắn
- Đánh giá độ phức tạp

38

Độ phức tạp (3)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Nội dung

- ▣ Khái niệm độ phức tạp
- ▣ Độ phức tạp: lý thuyết và thực tế
- ▣ Đánh giá độ phức tạp: ba trường hợp
- ▣ Các hàm tiệm cận
- ▣ Hệ thức truy hồi
- ▣ Độ phức tạp thực tế

Độ phức tạp

- Độ phức tạp
 - Độ phức tạp của một thuật toán là đo lường số các thao tác cơ bản mà thuật toán thực hiện trên một bộ dữ liệu vào
- Phụ thuộc vào dữ liệu vào
 - Độ phức tạp là một hàm phụ thuộc vào kích thước n của bộ dữ liệu vào
 - Độ phức tạp có ý nghĩa khi n lớn
- Đánh giá độ phức tạp thuật toán nhằm
 - Nghiên cứu hoạt động của thuật toán
 - Tối ưu hay không
 - Phát hiện những phần phức tạp, làm chậm thuật toán
 - So sánh các giải pháp khác nhau trong cùng ngữ cảnh
 - vấn đề
 - ngôn ngữ
 - máy tính

41

thời gian và bộ nhớ

- Đánh giá độ phức tạp
 - thời gian thực thi
 - bộ nhớ sử dụng
 - thường mâu thuẫn nhau
- Chúng ta thường tìm cách giảm độ phức tạp về mặt thời gian
 - bỏ qua độ phức tạp về mặt bộ nhớ
- Dẫn đến
 - thường làm tăng độ phức tạp về mặt bộ nhớ
 - làm tăng « độ phức tạp trí tuệ » của thuật toán
 - Thời gian phát triển
 - Rủi ro xảy ra lỗi
 - Chi phí bảo trì

42

lý thuyết và thực tế

- Hai phương pháp đánh giá độ phức tạp về mặt thời gian
 - Phương pháp lý thuyết
 - Phương pháp thực tế
- Phương pháp lý thuyết
 - đánh giá hoạt động của thuật toán khi kích thước dữ liệu n rất lớn
 - hoạt động của thuật toán được đánh giá bởi các hàm của n
 - không tính đến các chi tiết của thuật toán

43

lý thuyết và thực tế

- Phương pháp thực tế
 - Đánh giá một cách chi tiết thuật toán
 - Tất cả các câu lệnh đều được xem xét
 - Đánh giá riêng rẽ các hoạt động khác nhau
 - phép cộng/trừ số nguyên
 - phép nhân số nguyên
 - phép so sánh
 - thao tác đọc/ghi
 - truy cập bộ nhớ
 - ...
 - Mang lại các kết quả rất chi tiết
 - nhưng thường phức tạp
 - thường khó để so sánh (do đánh giá riêng rẽ)
 - Phát hiện các vùng phức tạp
 - Hỗ trợ tối ưu chương trình

44

lý thuyết và thực tế

- Quan hệ giữa phương pháp lý thuyết và phương pháp thực tế
 - Thường chỉ đánh giá tập các **thao tác cơ bản**
 - thời gian thực thi thuật toán tỷ lệ thuận với số các thao tác này
 - Ví dụ
 - Số phép so sánh và trao đổi dữ liệu đối với thuật toán sắp xếp
 - Số các phép cộng và phép nhân đối với thuật toán nhân hai ma trận
 - Số các truy cập đĩa đối với thuật toán thao tác trên CSDL
 - Các thao tác cơ bản được đánh giá riêng rẽ
 - Sự lựa chọn tập các thao tác cơ bản phụ thuộc
 - kích thước dữ liệu
 - chi tiết cần đánh giá
 - khả năng phân tích
 - ...

45

Tính độ phức tạp

- Dựa vào ngữ pháp của thuật toán, chúng ta có các nguyên tắc tính độ phức tạp như sau:
 - Đối với các thao tác cơ bản là các lệnh tuần tự thì cộng dồn số các thao tác
 - Đối với lệnh điều kiện, giả sử $P(X)$ là số thao tác cơ bản của cấu trúc lệnh X , thì:

$$P(\text{if } C \text{ then } I_1 \text{ else } I_2) \leq P(C) + \max(P(I_1), P(I_2))$$
 - Đối với các lệnh lặp, số các thao tác cơ bản trong vòng lặp là $\sum P(i)$, trong đó i biến điều khiển lặp, $P(i)$ số thao tác cơ bản ở lần lặp thứ i

46

Tính độ phức tạp

■ Đối với lời gọi hàm:

- Nếu không có hàm đệ quy, chúng ta luôn có thể tổ chức sao cho mỗi một hàm gọi hàm khác mà số thao tác cơ bản của hàm đó đã xác định
- Nếu là hàm đệ quy, tính số các thao tác cơ bản thường dẫn đến **hệ thức truy hồi**

■ Ví dụ

```
function factorial(n)
Begin
  If (n = 1) then
    factorial = 1
  Else
    factorial = n * factorial (n-1)
  EndIf
End
```

Thao tác cơ bản: số phép nhân

$$C(0) = 0$$

$$C(n) = C(n-1) + 1$$

$$\text{Vậy: } C(n) = n$$

47

Tính độ phức tạp: ba trường hợp

- Độ phức tạp phụ thuộc vào dữ liệu sử dụng bởi thuật toán
- Trường hợp tốt nhất
 - Dữ liệu được tổ chức sao cho thuật toán hoạt động hiệu quả nhất
 - Giá trị nhỏ nhất của độ phức tạp thực tế
- Trường hợp xấu nhất
 - Dữ liệu được tổ chức sao cho thuật toán hoạt động kém hiệu quả nhất
 - Giá trị lớn nhất của độ phức tạp thực tế
- Trường hợp trung bình
 - Dữ liệu tương ứng với sự tổ chức được xem là trung bình
 - Không đơn giản để xác định
 - Thường có ý nghĩa nhất
 - Không là trung bình của độ phức tạp tốt nhất và độ phức tạp xấu nhất

48

Tính độ phức tạp: ba trường hợp

- D_n : tập dữ liệu kích thước n
- $C_A(d)$: độ phức tạp của thuật toán A đối với dữ liệu d của tập D_n
- $p(d)$: xác suất xuất hiện của d

- Độ phức tạp trong trường hợp tốt nhất: $C_{\min_A}(n)$
 - $C_{\min_A}(n) = \min\{C_A(d), d \in D_n\}$
- Độ phức tạp trong trường hợp xấu nhất: $C_{\max_A}(n)$
 - $C_{\max_A}(n) = \max\{C_A(d), d \in D_n\}$
- Độ phức tạp trung bình: $C_{\text{avg}_A}(n)$
 - $C_{\text{avg}_A}(n) = \sum p(d) \cdot C_A(d)$

49

Tính độ phức tạp: ba trường hợp

- Nếu xác suất xuất hiện các dữ liệu là như nhau, thì
 - $\text{card}(D_n) = |D_n|$
 - $C_{\text{avg}_A}(n) = (1/|D_n|) \sum C_A(d)$
- Độ phức tạp trung bình nói chung là không thể xác định chính xác
- Điều chắc chắn
 - $C_{\min_A}(n) \leq C_{\text{avg}_A}(n) \leq C_{\max_A}(n)$

50

Tính độ phức tạp: ba trường hợp

▣ Ví dụ 1: nhân hai ma trận

```

matrix_product (A, B, C, n)
Begin
  For i from 1 to n
    For j from 1 to n
      C(i,j) = 0
      For k from 1 to n
        C(i,j) = C(i,j) + A(i,k) * B(k,j)
      EndFor
    EndFor
  EndFor
End

```

Quan hệ: $Cmin_A(n)$? $Cavg_A(n)$? $Cmax_A(n)$

51

Tính độ phức tạp: ba trường hợp

▣ Ví dụ 2: tìm kiếm tuần tự trong một danh sách

```

function lookup (L, X, n)
Begin
  i = 1
  While ((i ≤ n) and (L(i) ≠ X))
    i = i + 1
  EndWhile
  If (i > n) then
    i = 0
  EndIf
  lookup = i
End

```

Thao tác cơ bản: số các phép so sánh

- Trường hợp tốt nhất: $L(1) = X$, $Cmin_A(n) = 1$
- Trường hợp xấu nhất: X không có trong L hoặc $L(n) = X$, $Cmax_A(n) = n$

52

Tính độ phức tạp: ba trường hợp

□ Ví dụ 2: độ phức tạp trung bình

- q : xác suất X ở trong danh sách L
- X trong L : khả năng xuất hiện tại các vị trí như nhau
- $D_{n,i}$: tập dữ liệu với X ở vị trí i , $p(D_{n,i}) = q/n$
- $D_{n,0}$: tập dữ liệu với X không thuộc L , $p(D_{n,0}) = 1-q$
- $C_A(D_{n,i}) = i$ và $C_A(D_{n,0}) = n$

$$\begin{aligned} \text{Cavg}_A(n) &= \sum_{i=0}^n p(D_{n,i}) C_A(D_{n,i}) = (1-q)n + q/n \sum_{i=1}^n i \\ &= (1-q)n + q(n+1)/2 \end{aligned}$$

53

Tính độ phức tạp: ba trường hợp

□ Ví dụ 2: độ phức tạp trung bình

- Nếu $q = 1$ (X thuộc L), thì $\text{Cavg}_A(n) = (n+1)/2$
- Nếu $q = 1/2$ (X không thuộc L), thì $\text{Cavg}_A(n) = (3n+1)/4$
- Trường hợp trung bình
 - X nằm ở giữa danh sách L , tại vị trí $n/2$ hoặc $(n+1)/2$
 - độ phức tạp là $n/2$ hoặc $(n+1)/2$

54

Các hàm tiệm cận

- ▣ Trong trường hợp tổng quát, chúng ta thường không tính độ phức tạp chính xác, mà tính độ phức tạp tiệm cận (khi n rất lớn)
- ▣ Chúng ta xem xét các hàm (asymptotic notations) theo n và thay đổi của hàm khi mà n rất lớn
- ▣ Các hàm tiệm cận cho phép
 - đánh giá độ hiệu quả của thuật toán
 - so sánh hiệu quả của các thuật toán

55

Các hàm tiệm cận

- ▣ Kí hiệu o nhỏ
 - $f = o(g)$ khi $n \rightarrow \infty$ nếu:

$$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0, 0 \leq f(n) < cg(n)$$
 - $g(n)$ là tiệm cận trên của $f(n)$ với **mọi** hằng số c
 - $f(n)$ tăng chậm hơn $g(n)$:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$
 - Ví dụ
 - ▣ $n = o(n^2)$
 - ▣ $n^2 = o(n^4)$

56

Các hàm tiệm cận

□ Kí hiệu O lớn

- $f = O(g)$ khi $n \rightarrow \infty$ nếu:

$$\exists c > 0, \exists n_0 > 0, 0 \leq f(n) \leq cg(n), \forall n \geq n_0$$

- $g(n)$ là tiệm cận trên của $f(n)$ với **một số** hằng số c
 - Nếu $f = o(g)$ thì $f = O(g)$

- $f(n)$ không tăng nhanh hơn $g(n)$

- Ví dụ

- $2n = O(n^2)$
- $2n^2 = O(n^2)$, nhưng $2n^2 \neq o(n^2)$

57

Các hàm tiệm cận

□ Kí hiệu Θ

- $f = \Theta(g)$ khi $n \rightarrow \infty$ nếu:

$$\exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$$

- $f(n)$ và $g(n)$ tăng cùng tốc độ (so sánh trong nghĩa độ phức tạp)

- Ví dụ

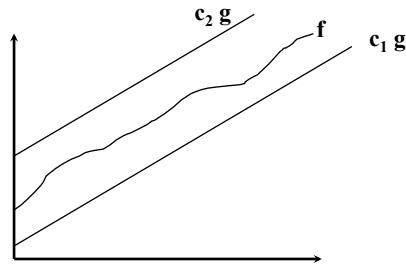
- $f(n) = n^2/2 - 3n = n^2(1/2 - 3/n)$, $g(n) = n^2$
- xác định c_1 và c_2 sao cho: $c_1 \leq 1/2 - 3/n \leq c_2$
- Với $c_2 = 1/2$, $n_0 = 12$, $c_1 = 1/2 - 3/12 = 1/4$
- Vậy $n^2/2 - 3n = \Theta(n^2)$

58

Các hàm tiệm cận

□ Kí hiệu Θ

- định nghĩa tiệm cận trên và tiệm cận dưới của $f(n)$



- Nếu $f = \Theta(g)$ thì:
 - $f = O(g)$
 - $g = O(f)$

59

Các hàm tiệm cận

□ Kí hiệu Ω

- $f = \Omega(g)$ khi $n \rightarrow \infty$ nếu:

$$\exists c > 0, \exists n_0 > 0, 0 \leq cg(n) \leq f(n), \forall n \geq n_0$$

- $g(n)$ là tiệm cận dưới của $f(n)$ với **một số** hằng số c
- $f(n)$ tăng ít nhất cũng nhanh bằng $g(n)$
- Ví dụ
 - $n^2 = \Omega(n)$

60

Các hàm tiệm cận

□ Các tính chất

- Nếu $f = \Theta(g)$ thì $f = O(g)$ và $f = \Omega(g)$
 - Suy ra từ định nghĩa

- Nếu $f = \Theta(g)$ thì $g = \Theta(f)$
 - Chứng minh

$$\exists(c_1, c_2, n_0), c_1 > 0, c_2 > 0, \forall n > n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$\exists(c_2, n_0), c_2 > 0, \forall n > n_0, 0 \leq \frac{1}{c_2} f(n) \leq g(n)$$

$$\exists(c_1, n_0), c_1 > 0, \forall n > n_0, 0 \leq g(n) \leq \frac{1}{c_1} f(n)$$

61

Các hàm tiệm cận

□ Hàm tương đương

- $f \sim g$ khi $n \rightarrow \infty$ nếu:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$$

- $f(n)$ và $g(n)$ tăng cùng tốc độ (một cách chính xác)
- $f(n)$ và $g(n)$ có cùng các giới hạn
- Ví dụ
 - $f(n) = n^2 - n$ và $g(n) = n^2$
 - $f(n) \sim g(n)$

62

Các hàm tiệm cận

▣ Độ phức tạp lý thuyết khi $n \rightarrow \infty$

$f = o(g)$	độ phức tạp của $f <$ độ phức tạp của g
$f = O(g)$	độ phức tạp của $f \leq$ độ phức tạp của g (theo các hằng số xác định)
$f = \Theta(g)$	độ phức tạp của f cùng cỡ độ phức tạp của g (theo các hằng số xác định)
$f = \Omega(g)$	độ phức tạp của $f \geq$ độ phức tạp của g (theo các hằng số xác định)
$f = \sim(g)$	độ phức tạp của $f =$ độ phức tạp của g

63

Các hàm tiệm cận

▣ Ví dụ 1

- Giả sử $C_A(n) = 8n^2 + 2n + 1$ và $C_B(n) = 1000n^2 + 10n + 2$ là các hàm biểu diễn thời gian thực thi của hai phương pháp cài đặt cùng một thuật toán trong trường hợp xấu nhất. Xác định O của hai hàm trên.
- $C_A(n) = 8n^2 + 2n + 1$
 $\leq 8n^2 + 2n^2 + n^2$ với $n \geq 1$
 $= 11n^2$
 chọn $c = 11, n_0 = 1$, thì $C_A(n) = O(n^2)$
- $C_B(n) = 1000n^2 + 10n + 2$
 $\leq 1000n^2 + 10n^2 + 2n^2$ với $n \geq 1$
 $= 1012n^2$
 chọn $c = 1012, n_0 = 1$, thì $C_B(n) = O(n^2)$
- Thuật toán: $O(n^2)$

64

Các hàm tiệm cận

□ Ví dụ 2

- Chứng minh rằng $\forall k \in \mathbb{N}$: $\sum_{i=1}^n i^k = \theta(n^{k+1})$
- Ta có: $\sum_{i=1}^n i^k \leq \sum_{i=1}^n n^k = n \cdot n^k = n^{k+1}$
- Vậy: $\sum_{i=1}^n i^k = O(n^{k+1})$ với $c = 1$
- Ta có: $\sum_{i=1}^n i^k \geq \sum_{i=\lfloor \frac{n}{2} \rfloor}^n i^k \geq \sum_{i=\lfloor \frac{n}{2} \rfloor}^n \left(\frac{n}{2}\right)^k \geq \frac{n}{2} \left(\frac{n}{2}\right)^k = \frac{n^{k+1}}{2^{k+1}}$
- Vậy: $\sum_{i=1}^n i^k = \Omega(n^{k+1})$ với $c = 1/2^{k+1}$

65

Hệ thức truy hồi

- Như đã trình bày, tính độ phức tạp hàm đệ quy thường dẫn đến hệ thức truy hồi
- Các họ hệ thức truy hồi thường gặp
 - Truy hồi bậc nhất
 - tuyến tính: $a_n = n a_{n-1} + g(n)$
 - không tuyến tính: $a_n = 1/(n + a_{n-1}) + g(n)$

hạng đầu tiên phải có giá trị
 - Truy hồi bậc hai
 - tuyến tính: $a_n = b a_{n-1} + c a_{n-2} + g(n)$
 - không tuyến tính: $a_n = 1/(a_{n-1} + a_{n-2}) + g(n)$

hai hạng đầu tiên phải có giá trị
 - Truy hồi bậc k
 - tuyến tính: $a_n = b_1 a_{n-1} + b_2 a_{n-2} + \dots + b_k a_{n-k} + g(n)$
 - tuyến tính thuần nhất hệ số hằng số: $a_n = b_1 a_{n-1} + b_2 a_{n-2} + \dots + b_k a_{n-k}$
 - không tuyến tính: ...

k hạng đầu tiên phải có giá trị

66

Hệ thức truy hồi

□ Các họ hệ thức truy hồi thường gặp (tiếp)

■ Truy hồi hoàn toàn

□ tuyến tính:

$$a_n = b_1 a_{n-1} + b_2 a_{n-2} + \dots + b_k a_{n-k} + \dots + b_{n-1} a_1 + g(n)$$

□ không tuyến tính: ...

n-1 hạng đầu tiên phải có giá trị

■ Truy hồi phân hoạch

$$a_n = b a_{\lfloor n/2 \rfloor} + c a_{\lceil n/2 \rceil} + g(n)$$

hoặc dưới dạng: $a_n = c a_{n/b} + g(n)$, với điều kiện n/b là số nguyên

■ Nhiều trường hợp, rất phức tạp để giải hệ thức truy hồi

■ Đề cập đến giải hệ thức

□ truy hồi tuyến tính bậc nhất

□ Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số

□ truy hồi phân hoạch

67

Hệ thức truy hồi

□ Truy hồi tuyến tính bậc nhất (1)

■ Dạng hệ số hằng số: $a_n = b a_{n-1} + g_n$, biết trước a_0

$$a_n = b a_{n-1} + g_n$$

$$a_{n-1} = b a_{n-2} + g_{n-1}$$

$$a_{n-2} = b a_{n-3} + g_{n-2}$$

...

$$a_1 = b a_0 + g_1$$

* với hệ số = b

* với hệ số = b^2

* với hệ số = b^{n-1}

$$a_n = b^n a_0 + \sum_{i=1}^n g_i b^{n-i}$$

68

Hệ thức truy hồi

□ Truy hồi tuyến tính bậc nhất (2)

- Dạng: $a_n = b_n a_{n-1} + g_n$, biết trước a_0

Đặt: $a_n = b_1 b_2 \dots b_n y_n$, $a_0 = y_0$

Vậy: $b_1 b_2 \dots b_n y_n = b_1 b_2 \dots b_n y_{n-1} + g_n$

Hay: $y_n = y_{n-1} + g_n / (b_1 b_2 \dots b_n)$

Dẫn đến: $y_n = y_0 + \sum_{i=1}^n \frac{g_i}{b_1 b_2 \dots b_i}$

Cuối cùng: $a_n = \prod_{i=1}^n b_i \left(a_0 + \sum_{i=1}^n \frac{g_i}{b_1 b_2 \dots b_i} \right)$

69

Hệ thức truy hồi

□ Truy hồi tuyến tính bậc nhất (3)

- Ví dụ tính:

- $a_n = 4 a_{n-1} - 1$, $a_0 = 1/3$

- $a_n = 4 a_{n-1} - 1$, $a_0 = 0.333$

- $a_n = 3 a_{n-1}^2$ và $a_0 = 1$
có thể đặt $b_n = \log_2 a_n$

70

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (1)
 - Dạng: $a_n = ba_{n-1} + ca_{n-2}$ (1)
 - Nghiệm của hệ thức trên có dạng $a_n = r^n$, r là hằng số
 - Nghĩa là: $r^n = br^{n-1} + cr^{n-2}$
 - Chia hai vế của hệ thức cho r^{n-2} , ta có:

$$r^2 = br + c$$
 - Khi đó $a_n = r^n$ là nghiệm của hệ thức truy hồi khi r là nghiệm của phương trình

$$r^2 - br - c = 0 \quad (2)$$

(gọi là phương trình đặc trưng của hệ thức truy hồi)
 - Trong trường hợp tổng quát, ta thu được hai nghiệm phân biệt r_1 và r_2
 - Khi đó, $a_n = \alpha r_1^n + \beta r_2^n$ cũng là nghiệm của (1), với α và β là các hằng số
 - Chứng minh ...

71

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (2)
 - $a_n = \alpha r_1^n + \beta r_2^n$
 - các hệ số α và β được xác định bởi các điều kiện đầu

$$\alpha + \beta = a_0$$

$$\alpha r_1 + \beta r_2 = a_1$$
 - Ví dụ
 - Tìm nghiệm của hệ thức truy hồi $a_n = a_{n-1} + 2a_{n-2}$ với các điều kiện đầu $a_0=2$ và $a_1=7$
 - Phương trình đặc trưng của hệ thức truy hồi: $r^2 - r - 2 = 0$
 - Nghiệm của phương trình đặc trưng: $r_1 = 2$ và $r_2 = -1$
 - Vậy $a_n = \alpha (2)^n + \beta (-1)^n$
 - Thay các điều kiện đầu, có

$$a_0 = \alpha + \beta = 2$$

$$a_1 = 2\alpha - \beta = 7$$
 - Vậy: $\alpha = 3, \beta = -1$
 - Nghiệm của hệ thức truy hồi: $a_n = 3 \cdot 2^n - (-1)^n$

72

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (3)
 - Nếu phương trình đặc trưng có nghiệm kép thì nghiệm của (1) là:

$$a_n = (\alpha n + \beta) r^n$$
 - Ví dụ
 - Tìm nghiệm của hệ thức truy hồi $a_n = 4a_{n-1} - 4a_{n-2}$ với các điều kiện đầu $a_0=3$ và $a_1=8$
 - Phương trình đặc trưng của hệ thức truy hồi: $r^2-4r+4=0$
 - Phương trình đặc trưng có nghiệm kép: $r = 2$
 - Vậy $a_n = (\alpha n + \beta)(2)^n$
 - Thay các điều kiện đầu, có

$$a_0 = 3 = \beta$$

$$a_1 = (\alpha + \beta)2 = 8$$
 - Vậy: $\alpha = 1, \beta = 3$
 - Nghiệm của hệ thức truy hồi: $a_n = n2^n + 3 \cdot 2^n$

73

Hệ thức truy hồi

- Truy hồi phân hoạch (1)
 - Truy hồi dạng: $a_n = b a_{\lfloor n/2 \rfloor} + c a_{\lceil n/2 \rceil} + g(n)$
 - Phân tích bài toán thành hai bài toán con
 - kích thước $\lfloor n/2 \rfloor$ và $\lceil n/2 \rceil$
 - vậy, thông thường $b = c (= 1)$
 - $g(n)$: chi phí phân tích và trộn các giải pháp con
 - Khi n lớn, có thể xem

$$\lfloor n/2 \rfloor = \lceil n/2 \rceil = n/2$$
 - Vậy, hệ thức truy hồi trở thành

$$a_n = 2 a_{n/2} + g(n)$$

74

Hệ thức truy hồi

□ Truy hồi phân hoạch (2)

- Thường được xét dưới dạng:

$$a_n = c a_{n/b} + g(n), \text{ với } n=b^k$$

- Hay $a_{b^k} = c a_{b^{k-1}} + g(b^k)$

- Đặt $t_k = a_{b^k}$

- Vậy ta có

$$\begin{aligned} t_k &= c t_{k-1} + g(b^k) \\ t_0 &= 1 (= a_1 = 1) \end{aligned}$$

$$\begin{aligned} t_k &= c t_{k-1} + g(b^k) \\ t_{k-1} &= c t_{k-2} + g(b^{k-1}) \\ &\dots \\ t_1 &= c t_0 + g(b^1) \end{aligned}$$

75

Hệ thức truy hồi

□ Truy hồi phân hoạch (3)

$$t_k = c t_{k-1} + g(b^k)$$

$$t_{k-1} = c t_{k-2} + g(b^{k-1})$$

$$t_{k-2} = c t_{k-3} + g(b^{k-2})$$

...

$$t_1 = c t_0 + g(b^1)$$

$$t_k = c^k + \sum_{i=0}^{k-1} c^i g(b^{k-i})$$

*c

*c²

$$*c^{k-1} \quad t_0=1$$

$$\text{với } n = b^k \Leftrightarrow k = \log_b n$$

$$b^{k-i} = n / b^i$$

$$c^{\log_b n} = n^{\log_b c}$$

$$a_n = n^{\log_b c} + \sum_{i=0}^{k-1} c^i g\left(\frac{n}{b^i}\right)$$

76

Hệ thức truy hồi

□ Ví dụ (1)

- Tính độ phức tạp của thuật toán sắp xếp trộn (merge sort)

```

merge-sort (A, p, q)
  Begin
    If (p < q) then
      r = (p+q)/2
      merge-sort (A, p, r)
      merge-sort (A, r + 1, q)
      merge (A, p, r, q)
    EndIf
  End

```

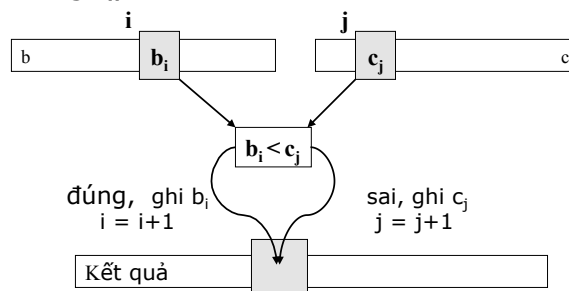
- Phân tích thành hai bài toán con có kích thước $\lfloor n/2 \rfloor$ và $\lfloor n/2 \rfloor$

77

Hệ thức truy hồi

□ Ví dụ (2)

- Trộn hai danh sách b và c kích thước $n/2$ đã được sắp xếp: merge()



- Nhiều nhất sử dụng $n-1$ phép so sánh

78

Hệ thức truy hồi

□ Ví dụ (3)

- Vậy ta có hệ thức truy hồi ($a_n = C(n)$)

$$a_n = 2 a_{n/2} + n - 1$$

$$a_n = n^{\log_b c} + \sum_{i=0}^{k-1} c^i g\left(\frac{n}{b^i}\right) \Rightarrow a_n = n^{\log_2 2} + \sum_{i=0}^{k-1} 2^i \left(\frac{n}{2^i} - 1\right)$$

sử dụng công thức: $\sum_{i=0}^n q^i = \frac{1-q^{n+1}}{1-q} \quad (q \neq 1)$

$$a_n = n + k n - \sum_{i=0}^{k-1} 2^i = n + k n - (2^k - 1) \quad \text{với } k = \log_2 n$$

$$a_n = n + n \log_2 n - (2^{\log_2 n} - 1) \quad \text{sử dụng } c^{\log_b n} = n^{\log_b c}$$

$$a_n = n \log_2 n + 1$$

$$\text{Vậy: } C(n) = \Theta(n \log(n))$$

79

Độ phức tạp thực tế

- Đánh giá một cách chi tiết thuật toán
- Tất cả các câu lệnh đều được xem xét
- Đánh giá riêng rẽ các loại thao tác khác nhau
 - phép cộng/trừ số nguyên
 - phép nhân số nguyên
 - phép so sánh
 - thao tác đọc/ghi
 - truy cập bộ nhớ
 - ...
- Mang lại các kết quả rất chi tiết
 - nhưng thường phức tạp
 - thường khó để so sánh (do đánh giá riêng rẽ)

80

Độ phức tạp thực tế

□ Ví dụ (1)

- Đánh giá độ phức tạp của thuật toán sắp xếp chèn (insertion sort)
- Mỗi lệnh được gán chi phí (cost) về thời gian thực thi và số lần thực thi (times). Với mỗi $j = 2, 3, \dots, n$, kí hiệu t_j là số lần thực thi lệnh lặp While

insertion-sort(A)	cost	times
<u>Begin</u>		
For j from 2 to n	c_1	n
key = A[j]	c_2	$n-1$
i = j - 1	c_3	$n-1$
While (i > 0 and A[i] > key) do	c_4	$\sum_{j=2}^n t_j$
A[i+1] = A[i]	c_5	$\sum_{j=2}^n (t_j - 1)$
i = i - 1	c_6	$\sum_{j=2}^n (t_j - 1)$
EndWhile		
A[i+1] = key	c_7	$n-1$
EndFor		
<u>End</u>		

81

Độ phức tạp thực tế

□ Ví dụ (2)

- Tổng thời gian thực thi là

$$C(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7 (n-1)$$

- Độ phức tạp trong trường hợp tốt nhất: danh sách đã được sắp xếp đúng thứ tự, khi đó $t_j = 1$ với mọi j . Vậy:

$$\begin{aligned} C(n) &= c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 (n-1) + c_7 (n-1) \\ &= n(c_1 + c_2 + c_3 + c_4 + c_7) - (c_2 + c_3 + c_4 + c_7) \end{aligned}$$

Có thể viết $C(n) = an + b$, với a và b là các hằng số
Độ phức tạp trong trường hợp tốt nhất là hàm tuyến tính n

82

Độ phức tạp thực tế

□ Ví dụ (2)

- Độ phức tạp trong trường hợp xấu nhất: danh sách đã được sắp xếp theo thứ tự ngược lại, khi đó $t_j = j$ với mọi j .

- Ta có:

- Vậy: $\sum_{j=1}^n j = \frac{n(n+1)}{2}$, $\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$, $\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$

$$C(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

Có thể viết $C(n) = an^2 + bn + c$, với a , b và c là các hằng số
Độ phức tạp trong trường hợp xấu nhất là hàm bình phương n

83

Độ phức tạp thực tế

□ Ví dụ (3)

- Độ phức tạp trong trường hợp trung bình

- giả sử áp dụng thuật toán cho danh sách n phần tử được chọn một cách ngẫu nhiên.
- Giá trị của t_j ? Phần tử $A[j]$ được chèn vào vị trí nào trong danh sách $A[1..j-1]$?
- Trường hợp trung bình, một nửa số phần tử của $A[1..j-1]$ lớn hơn $A[j]$ và một nửa số phần tử của $A[1..j-1]$ nhỏ hơn $A[j]$
- Vậy $t_j = j/2$

- Tương tự trường hợp xấu nhất, $C(n)$ sẽ có dạng $C(n) = an^2 + bn + c$, với a , b và c là các hằng số
- Độ phức tạp trong trường hợp trung bình là hàm bình phương n

84

Độ phức tạp thực tế

□ Ví dụ (4)

- Tuy nhiên, thông thường cái chúng ta quan tâm thực sự là giá trị của các hàm tiệm cận (thường là O và Θ)
- Chỉ có số hạng quan trọng của biểu thức độ phức tạp có ý nghĩa khi n rất lớn, bỏ qua các số hạng còn lại
- Khi n lớn, chúng ta bỏ qua luôn hằng số của số hạng quan trọng
- Vậy độ phức tạp của thuật toán sắp xếp chèn trong
 - trường hợp tốt nhất: $\Theta(n)$
 - trường hợp xấu nhất: $\Theta(n^2)$
 - trường hợp trung bình: $\Theta(n^2)$

85

Các lớp thuật toán theo độ phức tạp

- **Các thuật toán thường được so sánh bởi độ phức tạp trong trường hợp xấu nhất**, tức là giá trị của $O(g(n))$
- Các lớp thuật toán theo độ phức tạp
 - Hằng số: $O(1)$, tất cả các lệnh thực thi đúng một lần không phụ thuộc kích thước dữ liệu
 - Dưới tuyến tính: $O(\log(n))$, kích thước của bài toán được chia nhỏ bởi một hằng số ở mỗi bước lặp
 - Tuyến tính:
 - $O(n)$, vòng lặp n lần, thân vòng lặp thực hiện công việc độc lập với n
 - $O(n \log(n))$, kích thước của bài toán được chia nhỏ bởi một hằng số ở mỗi bước lặp, và ở mỗi lần chia nhỏ một bước duyệt tuyến tính các dữ liệu được thực hiện
 - Đa thức: $O(n^k)$, các thuật toán được xem là chậm khi $k > 3$
 - Hàm mũ: $O(k^n)$, $k \geq 2$, các thuật toán là không thể ứng dụng được
 - Các **thuật toán sử dụng được** thường có độ phức tạp $\leq O(n^3)$

86

Các lớp thuật toán theo độ phức tạp

▣ Minh họa độ phức tạp các lớp thuật toán

n	$\log_2(n)$	$n\log_2(n)$	n^2	n^3	2^n
1	0	0	1	1	2
10	3.32	33.2	100	10^3	1024
100	6.64	664	10^4	10^6	1.26^{30}
1000	9.965	9 965	10^6	10^9	∞
10^6	19.931	19 931 568	10^{12}	10^{18}	∞

87

Bài tập

▣ Bài 1

- Giả sử hàm $f(n) = \log(2n + \alpha)$ được định nghĩa, chứng minh rằng: $f(n) = \Theta(\log(n))$

▣ Bài 2

- Giả sử có một danh sách n phần tử có giá trị khác nhau. Hãy
 1. Xây dựng thuật toán xác định phần tử nhỏ nhất và phần tử lớn nhất
 2. Đánh giá độ phức tạp của thuật toán bởi số phép so sánh
 3. Đề xuất thuật toán hiệu quả hơn
 4. Đánh giá độ phức tạp của thuật toán bởi số phép so sánh

88

Bài tập

□ Bài 3

- Đánh giá độ phức tạp của thuật toán sau

```
timkiemnhiphan (A, x, l, r)
// tìm x trong danh sách A[l,r]
begin
  if (l = r) then return (l)
  else
    m = (l + r)/2
    if (x ≤ A[m]) then return (timkiemnhiphan (A, x, l, m))
    else return (timkiemnhiphan (A, x, m+1, r))
  endif
endif
end
```

89

Đệ quy (4)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường Đại học Bách khoa
Đại học Đà Nẵng

Đệ quy

- Thuật toán được gọi là đệ quy khi nó được xây dựng dựa trên chính nó
- Đơn đệ quy (simple recursion)
 - Chẳng hạn, định nghĩa hàm tính x^n
 - Hàm được định nghĩa đệ quy

$$x^n = \begin{cases} 1 & \text{ khi } n = 0 \\ x \times x^{n-1} & \text{ khi } n \geq 1 \end{cases}$$

- Thuật toán đệ quy

```
function ham_mu (x, n)
begin
    if (n=0) then ham_mu = 1
    else ham_mu = x * ham_mu(x, n-1)
    endif
end
```

91

Đệ quy

- Đa đệ quy (multiple recursion)
 - Một định nghĩa đệ quy có thể có nhiều hơn một lời gọi đệ quy
 - Ví dụ:
 - Định nghĩa dãy số Fibonacci

$$F_0 = 1, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$
 - Thuật toán

```
function fib (n)
begin
    if ((n=0) or (n=1)) then fib = 1
    else fib = fib(n-1) + fib(n-2)
    endif
end
```

92

Đệ quy

□ Đệ quy chéo (mutual recursion)

- Các định nghĩa được gọi là đệ quy chéo nếu chúng phụ thuộc lẫn nhau

- Ví dụ

- Định nghĩa số chẵn/lẻ

$$\text{even}(n) = \begin{cases} \text{true} & \text{if } n = 0 \\ \text{odd}(n-1) & \text{else} \end{cases} \quad \text{odd}(n) = \begin{cases} \text{false} & \text{if } n = 0 \\ \text{even}(n-1) & \text{else} \end{cases}$$

- Thuật toán

```
function even (n)
begin
  if (n=0) then even = true
  else even = odd(n-1)
  endif
end
```

```
function odd (n)
begin
  if (n=0) then odd = false
  else odd = even(n-1)
  endif
end
```

Đệ quy

□ Đệ quy chồng (implicated recursion)

- Các định nghĩa được gọi đệ quy lồng nhau

- Ví dụ

- Định nghĩa hàm Ackermann

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0, n = 0 \\ A(m-1, A(m, n-1)) & \text{else} \end{cases}$$

- Thuật toán

```
function Ackermann (m, n)
begin
  if (m=0) then Ackermann = n+1
  else if (n=0) Ackermann = Ackermann(m-1, 1)
  else Ackermann = Ackermann(m-1, Ackermann(m, n-1))
  endif
endif
end
```

Đệ quy

□ Nguyên tắc

- Chúng ta cần có
 - một số trường hợp mà giải pháp xác định – « trường hợp đơn giản »: các trường hợp dừng của đệ quy
 - một cách để chuyển từ một « trường hợp phức tạp » thành « trường hợp đơn giản »

□ Khó khăn

- Cần bảo đảm rằng, đệ quy sẽ dừng khi gặp giải pháp đã biết
 - Hàm phải được định nghĩa trên toàn miền dữ liệu

□ Giải pháp

- Dãy các giá trị liên nhau của các tham số được gọi phải thay đổi đơn điệu và đạt đến một giá trị mà giải pháp tương ứng đã được xác định

95

Đệ quy

□ Ví dụ 1

- Thuật toán sau kiểm tra a có là ước số của b

```
function divisor (a, b) // giả sử a>0, b>0
begin
  if (a≥b) then
    if (a=b) divisor = true
    else divisor = false
    endif
  else divisor=divisor(a, b-a)
  endif
end
```

- Dãy các giá trị b, b-a, b-2a ... liên tục giảm cho đến khi $a \geq b$ thì sẽ dừng, trường hợp đã được xác định

96

Đệ quy

□ Ví dụ 2

■ Thuật toán

```
function syracuse (n)
begin
  if (n=0 or n=1) then syracuse = 1
  else
    if (n mod 2 = 0) syracuse = syracuse(n/2)
    else syracuse = syracuse(3*n+1)
  endif
end
```

- Thuật toán được định nghĩa rõ ràng
- Thuật toán có dừng ?

97

Đệ quy

□ Không thể xác định tính dừng (1)

■ Vấn đề

- Có thể xây dựng công cụ tự động kiểm tra một thuật toán P có thể dừng khi thực thi trên bộ dữ liệu D?
- Vào
 - thuật toán P
 - bộ dữ liệu D
- Ra
 - đúng, nếu thuật toán P dừng trên bộ dữ liệu D
 - sai, nếu ngược lại

98

Đệ quy

□ Không thể xác định tính dừng (2)

- Giả sử tồn tại chương trình *terminate* kiểm tra tự động tính dừng của một thuật toán
- Từ chương trình *terminate* chúng ta xây dựng chương trình sau

```

program Q
begin
    result = terminate(Q)
    while (result = true)
        wait(1 minute)
    endwhile
end

```

➡ Vấn đề dừng là không thể xác định !

99

Đệ quy

□ Thứ tự của lời gọi đệ quy

- Hãy cho biết kết quả của hai thuật toán sau

```

T(n) // n ≥ 0
begin
    if (n=0) then do nothing
    else
        T(n-1)
        print(n) //in n
    endif
end

```

```

G(n) // n ≥ 0
begin
    if (n=0) then do nothing
    else
        print(n) //in n
        G(n-1)
    endif
end

```

- Thuật toán đệ quy in dãy nhị phân tương ứng của một số nguyên

100

Đệ quy

□ Ví dụ thuật toán đệ quy (1)

- Tháp Hà Nội: có 3 cọc A, B và C, mỗi cọc có thể chồng các đĩa có kích thước khác nhau, nguyên tắc chồng đĩa to dưới đĩa nhỏ trên; yêu cầu chuyển n đĩa trên cọc A sang cọc C với các điều kiện:
 - Mỗi lần chỉ được chuyển một đĩa
 - Không khi nào có tình huống đĩa to chồng trên đĩa nhỏ
 - Được sử dụng cọc B làm cọc trung gian khi chuyển đĩa

101

Đệ quy

□ Ví dụ thuật toán đệ quy (2)

- Giả thiết
 - chúng ta giải quyết được bài toán với n-1 đĩa
- Nguyên tắc
 - Để chuyển n đĩa từ cọc A sang cọc C, thực hiện
 - chuyển n-1 đĩa nhỏ hơn từ cọc A sang cọc B
 - chuyển đĩa lớn nhất từ cọc A sang cọc C
 - chuyển n-1 đĩa nhỏ hơn từ cọc B sang cọc C
- Thuật toán

```
Hanoi(n, A, B, C)
begin
  if (n=1) then chuyển đĩa lớn từ cọc A sang cọc C
  else Hanoi(n-1, A, C, B)
        chuyển đĩa lớn từ cọc A sang cọc C
        Hanoi(n-1, B, A, C)
  endif
end
```

Đệ quy

▣ Ví dụ thuật toán đệ quy (3)

- Tính độ phức tạp
 - ▣ Tính số lần chuyển đĩa

$$C(n) = \begin{cases} 1 & \text{if } n = 1 \\ C(n-1) + 1 + C(n-1) & \text{else} \end{cases}$$

$$C(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2C(n-1) + 1 & \text{else} \end{cases}$$

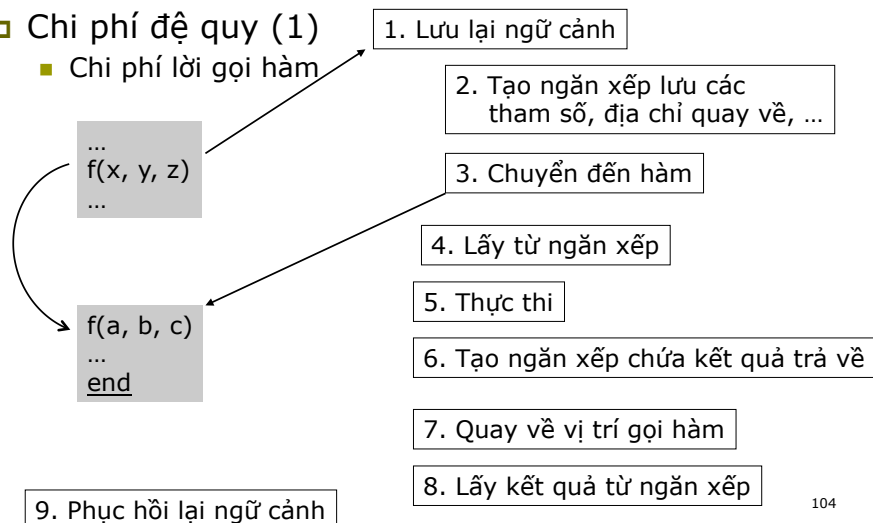
$$C(n) = 2^n - 1$$

103

Đệ quy

▣ Chi phí đệ quy (1)

- Chi phí lời gọi hàm



104

Đệ quy

Chi phí đệ quy (2)

Ví dụ

```
factorial(n)
begin
  factorial = 1
  for i = 2 to n
    factorial = factorial * i
  endfor
end
```

n-1	phép nhân
n	phép gán
n	phép gán (vòng lặp)
n-1	phép tăng 1 (vòng lặp)
n	phép so sánh

```
factorial(n)
begin
  if (n = 1) then
    factorial = 1
  else
    factorial = n * factorial(n-1)
  endif
end
```

n-1	phép nhân
n	phép gán
n-1	phép trừ (tính n-1)
n	phép so sánh
n	lời gọi hàm

➔ Chi phí đệ quy rất lớn do lời gọi hàm

105

Khử đệ quy

Chuyển thuật toán đệ quy thành thuật toán tương đương không chứa lời gọi đệ quy

Sử dụng vòng lặp

Hai trường hợp đệ quy

Đệ quy kết thúc (tail recursion)

- Thuật toán được gọi là đệ quy kết thúc nếu nó không chứa bất kỳ xử lý nào sau lời gọi đệ quy

Đệ quy không kết thúc (non tail recursion)

- Thuật toán được gọi là đệ quy không kết thúc nếu nó chứa các xử lý sau lời gọi đệ quy

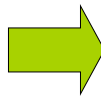
106

Khử đệ quy

▣ Đệ quy kết thúc (tail-recursion)

- Sơ đồ tổng quát của thuật toán đệ quy kết thúc

```
P(U)
begin
  if C then
    D
    P( $\alpha(U)$ )
  else
    T
  endif
end
```



```
P'(U)
begin
  while C do
    D
    U =  $\alpha(U)$ 
  endwhile
  T
end
```

U : danh sách các tham số
 C : điều kiện phụ thuộc U
 D : xử lý cơ bản của thuật toán
 $\alpha(U)$: biểu diễn sự chuyển đổi tham số
 T : xử lý dừng

107

Khử đệ quy

▣ Đệ quy kết thúc (2)

- Ví dụ: khử đệ quy của thuật toán sau

```
bsearch(X, A, l, r)
begin
  if (l ≤ r) then
    m = (l+r)/2
    if (X = A[m]) then bsearch = m
    else if (X < A[m]) then bsearch = bsearch(X, A, l, m-1)
    else bsearch = bsearch(X, A, m+1, r)
    endif
  endif
else
  bsearch = 0
endif
end
```

108

Khử đệ quy

▣ Đệ quy kết thúc (3)

- Ví dụ: thuật toán lặp tương đương

```

bsearch' (X, A)
begin
  l = 1
  r = n
  while (l ≤ r) do
    m = (l+r)/2
    if (X = A[m]) then bsearch' = m; return
    else if (X < A[m]) then r = m-1
        else l = m+1
    endif
  endwhile
  bsearch' = 0
end

```

109

Khử đệ quy

▣ Đệ quy không kết thúc (non tail-recursion)

- Cần ghi nhớ lại *ngữ cảnh* của lời gọi đệ quy
 - ▣ điển hình là các *tham số* của lời gọi đệ quy
- Sử dụng cấu trúc *ngăn xếp* (stack) để ghi nhớ ngữ cảnh
 - ▣ Các thao tác với ngăn xếp
 - create
 - isempty
 - push
 - pop
 - top
- Hai cách khử đệ quy không kết thúc

110

Khử đệ quy

▣ Đệ quy không kết thúc (2)

■ Cách 1

```

Q(U)
begin
  if C(U) then
    B(U)
    Q( $\alpha(U)$ )
    E(U)
  else
    T(U)
  endif
end

```



```

Q'(U)
begin
  create(S)
  while C(U) do
    B(U)
    push(S, U)
    U =  $\alpha(U)$ 
  endwhile
  T(U)
  while not isempty(S) do
    U = top(S)
    E(U)
    pop(S)
  endwhile
end

```

111

Khử đệ quy

▣ Đệ quy không kết thúc (3)

■ Cách 1

▣ Minh họa

```

Gọi Q( $U_0$ )
C( $U_0$ ) đúng
B( $U_0$ )
Gọi Q( $\alpha(U_0)$ )
C( $\alpha(U_0)$ ) đúng
B( $\alpha(U_0)$ )
Gọi Q( $\alpha(\alpha(U_0))$ )
C( $\alpha(\alpha(U_0))$ ) sai
T( $\alpha(\alpha(U_0))$ )
E( $\alpha(U_0)$ )
E( $U_0$ )

```



Gọi Q'(U_0) ?

112

Khử đệ quy

□ Đệ quy không kết thúc (4)

- Cách 1
- Ví dụ

```
T(n) // n ≥ 0
begin
  if (n=0) then do nothing
  else
    T(n-1)
  endif
end
  print(n) // in n
```



```
T'(n) // n ≥ 0
begin
  create(S)
  if (n=0) then do nothing
  else
    while (n>0) do
      push(S, n)
      n = n-1
    endwhile
    while (not isempty(S)) do
      n = top(S)
      print(n) // in n
      pop(S)
    endwhile
  endif
end
```

Khử đệ quy

□ Đệ quy không kết thúc (5)

- Cách 2

```
Q(U)
begin
  if C(U) then
    B(U)
    Q(α(U))
    E(U)
  else
    T(U)
  endif
end
```



```
Q'(U)
begin
  create(S)
  push(S, (newcall, U))
  while not isempty(S) do
    (state, V) = top(S)
    pop(S)
    if (state = newcall) then
      U = V
      if C(U) then
        B(U)
        push(S, (end, U))
        push(S, (newcall, α(U)))
      else T(U)
      endif
    endif
    if (state = end) then
      U = V
      E(U)
    endif
  endwhile
end
```

Khử đệ quy

▣ Đệ quy không kết thúc (6)

- Cách 2
 - ▣ Minh họa

```
Gọi Q(U0)
C(U0) đúng
B(U0)
Gọi Q(α(U0))
C(α(U0)) đúng
B(α(U0))
Gọi Q(α(α(U0)))
C(α(α(U0))) sai
T(α(α(U0)))
E(α(U0))
E(U0)
```



Gọi Q' (U₀) ?

115

Khử đệ quy

▣ Đệ quy không kết thúc (7)

- Cách 2
 - ▣ Ví dụ

```
T(n) // n ≥ 0
begin
  if (n=0) then do nothing
  else
    T(n-1)
    print(n) // in n
  endif
end
```



```
T'(n)
begin
  create(S)
  push(S, (newcall, n))
  while not isempty(S) do
    (state, k) = top(S)
    pop(S)
    if (state = newcall) then
      if (k > 0) then
        push(S, (end, k))
        push(S, (newcall, k-1))
      else do nothing
    endif
  endif
  if (state = end) then
    print(k)
  endif
endwhile
end
```

Khử đệ quy

- ▣ Thuật toán sử dụng vòng lặp thường hiệu quả hơn
- ▣ Thuật toán đệ quy thường dễ xây dựng hơn
- ▣ Phần lớn các trình biên dịch có thể tự động khử đệ quy kết thúc
- ▣ Luôn có thể khử đệ quy của một thuật toán

117

Bài tập

▣ Bài tập (1)

■ Bài 1

- ▣ Định nghĩa dãy số Fibonacci

$$\text{Fib}_0 = 1, \text{Fib}_1 = 1$$

$$\text{Fib}_n = \text{Fib}_{n-1} + \text{Fib}_{n-2}$$

- ▣ Hãy thực hiện

1. Xây dựng thuật toán đệ quy tính $\text{Fib}(n)$
2. Chứng minh rằng độ phức tạp (bởi số phép cộng) của thuật toán là $\Omega(2^{n/2})$
3. Xây dựng thuật toán tính cặp $(\text{Fib}(n), \text{Fib}(n-1))$ với $n > 0$
4. Sử dụng thuật toán trong câu 3 để xây dựng thuật toán mới tính $\text{Fib}(n)$
5. Đánh giá độ phức tạp (bởi số phép cộng) của thuật toán trên

118

Bài tập

□ Bài tập (2)

■ Bài 2

Ước số chung lớn nhất của hai số nguyên dương được định nghĩa như sau

- nếu $x = y$ thì $usc(x, y) = x$
- nếu $x > y$ thì $usc(x, y) = usc(x-y, y)$
- nếu $x < y$ thì $usc(x, y) = usc(x, y-x)$

1. Xây dựng thuật toán đệ quy tính ước số chung lớn nhất hai số nguyên dương
2. Khử đệ quy của thuật toán

■ Bài 3

1. Xây dựng thuật toán đệ quy in dãy nhị phân tương ứng của một số nguyên
2. Khử đệ quy thuật toán trên

119

Chia đề' trị (5)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Chia để trị (Divide and Conquer)

□ Nguyên tắc

- Nhiều thuật toán có cấu trúc đệ quy
 - Để giải quyết vấn đề đặt ra, thuật toán gọi lại chính nó để giải quyết các vấn đề con có kích thước nhỏ hơn, cuối cùng kết hợp các kết quả thu được giải pháp
- Gồm các bước
 - **Chia**: chia vấn đề thành các vấn đề con
 - **Trị**: giải quyết các vấn đề con một cách đệ quy, nếu vấn đề con có kích thước đủ nhỏ thì giải quyết trực tiếp
 - **Kết hợp**: các kết quả của các vấn đề con là giải pháp cho vấn đề đặt ra

121

Chia để trị

□ Cấu trúc chung

```

chia-để-trị(x: bài toán) : giải pháp
begin
  if (x nhỏ và đơn giản) then
    return (giải pháp)
  else
    phân tích x thành nhiều bài toán con  $x_1, x_2, \dots, x_k$ 
    for i from 1 to k do
       $y_i = \text{chia-để-trị}(x_i)$ 
    endfor
    kết hợp các giải pháp  $y_i$  thành giải pháp y của x
    return (y)
  endif
end
  
```

122

Một số ứng dụng

- ▣ Tìm giá trị lớn nhất và giá trị nhỏ nhất
- ▣ Nhân hai ma trận
- ▣ Quicksort
- ▣ Chọn phần tử
- ▣ Tính bao đóng lỗi

123

Tìm giá trị lớn nhất và giá trị nhỏ nhất

- ▣ Bài toán
 - Tìm giá trị lớn nhất (max) và giá trị nhỏ nhất (min) trong một danh sách $A[1..n]$. Cần sử dụng bao nhiêu phép so sánh giữa các phần tử của A ?
- ▣ Thuật toán vét cạn

```

maxmin(A, n)
begin
  max = A[1]
  min = A[1]
  for i from 2 to n
    if (A[i] > max) then max = A[i]
    endif
    if (A[i] < min) then min = A[i]
    endif
  endfor
  return (max, min)
end

```

124

Tìm giá trị lớn nhất và giá trị nhỏ nhất

▣ Thuật toán chia để trị

```

maxmin(A,x,y)
begin
  if (y-x ≤ 1) then
    return (max(A[x], A[y]), min(A[x], A[y]))
  else
    (max1, min1) = maxmin(A, x, [(x+y)/2])
    (max2, min2) = maxmin(A, [(x+y)/2]+1, y)
    return (max(max1, max2), min(min1, min2))
  endif
end

```

125

Tìm giá trị lớn nhất và giá trị nhỏ nhất

▣ Phân tích thuật toán

■ Tính số phép so sánh

- $C(n)$: số phép so sánh, với $n = y - x + 1$
- Giả sử n lũy thừa của 2, nghĩa là $y - x$ lẻ và $x+y$ cũng lẻ
- Kích thước các vấn đề con

$$\lfloor (x+y)/2 \rfloor - x + 1 = \frac{x+y-1}{2} - x + 1 = \frac{y-x+1}{2} = \frac{n}{2}$$

$$y - (\lfloor (x+y)/2 \rfloor + 1) + 1 = y - \frac{x+y-1}{2} = \frac{y-x+1}{2} = \frac{n}{2}$$

- Vậy

$$C(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2C(n/2) + 2 & \text{else} \end{cases}$$

126

Tìm giá trị lớn nhất và giá trị nhỏ nhất

- Phân tích thuật toán
 - Tính số phép so sánh

$$\begin{aligned}
 C(n) &= 2C(n/2) + 2 \\
 &= 2^2 C(n/4) + 2^2 + 2 \\
 &= 2^3 C(n/8) + 2^3 + 2^2 + 2 \\
 &= 2^i C(n/2^i) + \sum_{k=1}^i 2^k = 2^{\log n - 1} C(2) + \sum_{k=1}^{\log n - 1} 2^k \\
 &= 2^{\log n - 1} + \frac{1 - 2^{\log n - 1 + 1}}{1 - 2} = \frac{2^{\log n}}{2} + 2^{\log n} - 1 = \frac{3}{2}n - 1
 \end{aligned}$$

→ Thuật toán chia để trị chỉ sử dụng 75% số phép toán so sánh so với thuật toán vét cạn

127

Nhân hai ma trận

- Bài toán
 - Nhân hai ma trận vuông có n phần tử: $C = A.B$
- Thuật toán vét cạn

```

matrixproduct (A, B, n)
begin
  for i from 1 to n
    for j from 1 to n
      C(i,j) = 0
      for k from 1 to n
        C(i,j) = C(i,j) + A(i,k) * B(k,j)
      endfor
    endfor
  endfor
  return (C)
end

```

Thuật toán thực hiện $O(n^3)$ phép cộng và phép nhân ¹²⁸

Nhân hai ma trận

▣ Thuật toán chia để trị (1)

- Giả sử $n = 2^k$
- Chia các ma trận A, B, C thành các ma trận có kích thước $n/2$, khi đó $C = A.B$ tương ứng

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

- Từ đó, ta có
 - $r = ae + bg$
 - $s = af + bh$
 - $t = ce + dg$
 - $u = cf + dh$

129

Nhân hai ma trận

▣ Thuật toán chia để trị (2)

```

matrixproduct (A, B, n)
begin
  if (n = 1) then return (A.B)
  else
    Chia A và B thành 8 ma trận kích thước n/2: a, b, ..., h
    r = matrixproduct(a, e, n/2) + matrixproduct(b, g, n/2)
    s = matrixproduct(a, f, n/2) + matrixproduct(b, h, n/2)
    t = matrixproduct(c, e, n/2) + matrixproduct(d, g, n/2)
    u = matrixproduct(c, f, n/2) + matrixproduct(d, h, n/2)
    Ghép r, s, t và u có ma trận C
    return C
  endif
end

```

Độ phức tạp: $C(n) = 8C(n/2) + n^2$, $C(1) = 1$
 trong đó, n^2 là số phép cộng

130

Nhân hai ma trận

▣ Thuật toán chia để trị (3)

$$\begin{aligned}
 C(n) &= 8C(n/2) + n^2 \\
 &= 8(8C(n/4) + (n/2)^2) + n^2 \\
 &= 8^2 C(n/4) + 2n^2 + n^2 \\
 &= 8^i C(n/2^i) + n^2 \sum_{k=0}^{i-1} 2^k = 8^{\log n} C(1) + n^2 \sum_{k=0}^{\log n - 1} 2^k \\
 &= 8^{\log n} + n^2 \frac{1 - 2^{\log n - 1 + 1}}{1 - 2} = 8^{\log n} + n^2 (2^{\log n} - 1) = \\
 &= 2^{3 \log n} + n^2 (n - 1) = n^3 + n^2 (n - 1) = 2n^3 - n^2
 \end{aligned}$$

➡ $O(n^3)$

131

Nhân hai ma trận

▣ Thuật toán Strassen (1)

$$\begin{aligned}
 m_1 &= (a+c)(e+f) \\
 m_2 &= (b+d)(g+h) \\
 m_3 &= (a-d)(e+h) \\
 m_4 &= a(f-h) \\
 m_5 &= (c+d)e \\
 m_6 &= (a+b)h \\
 m_7 &= d(g-e)
 \end{aligned}$$

Chỉ thực hiện 7 phép nhân ma trận so với 8 phép nhân ma trận của thuật toán chia để trị

Thực hiện nhiều phép cộng và trừ ma trận hơn thuật toán chia để trị

▣ Khi đó

$$\begin{aligned}
 r &= m_2 + m_3 - m_6 - m_7 \\
 s &= m_4 + m_6 \\
 t &= m_5 + m_7 \\
 u &= m_1 - m_3 - m_4 - m_5
 \end{aligned}$$

132

Nhân hai ma trận

□ Thuật toán Strassen (2)

■ Tại sao đúng

$$\begin{aligned}
 r &= m_2 + m_3 - m_6 - m_7 \\
 &= (b+d)(g+h) + (a-d)(e+h) - (a+b)h - d(g-e) \\
 &= bg + bh + dg + dh + ae + ah - de - dh - ah - bh - dg + de \\
 &= ae + bg
 \end{aligned}$$

$$s = m_4 + m_6 = a(f-h) + (a+b)h = af + bh$$

$$t = m_5 + m_7 = (c+d)e + d(g-e) = ce + dg$$

$$\begin{aligned}
 u &= m_1 - m_3 - m_4 - m_5 \\
 &= (a+c)(e+f) - (a-d)(e+h) - a(f-h) - (c+d)e \\
 &= ae + af + ce + cf - ae - ah + de + dh - af + ah - ce - de \\
 &= cf + dh
 \end{aligned}$$

133

Nhân hai ma trận

□ Thuật toán Strassen (3)

```

matrixproduct (A, B, n)
begin
  if (n = 1) then return (A.B)
  else
    Chia A và B thành 8 ma trận kích thước n/2: a, b, ..., h
    m1 = matrixproduct(a+c, e+f, n/2)
    m2 = matrixproduct(b+d, g+h, n/2)
    m3 = matrixproduct(a-d, e+h, n/2)
    m4 = matrixproduct(a, f-h, n/2)
    m5 = matrixproduct(c+d, e, n/2)
    m6 = matrixproduct(a+b, h, n/2)
    m7 = matrixproduct(d, g-e, n/2)
    r = m2 + m3 - m6 - m7
    s = m4 + m6
    t = m5 + m7
    u = m1 - m3 - m4 - m5
  endif
end
  
```

134

Nhân hai ma trận

□ Thuật toán Strassen (4)

■ Độ phức tạp

$$C(n) = 7C(n/2) + 18n^2/4$$

$$C(1) = 1$$

135

Nhân hai ma trận

□ Thuật toán Strassen (5)

■ Độ phức tạp

$$\begin{aligned}
 C(n) &= 7C(n/2) + \frac{9}{2}n^2 \\
 &= 7(7C(n/4) + \frac{9}{2}(n/2)^2) + \frac{9}{2}n^2 \\
 &= 7^2C(n/8) + \frac{9}{2}7n^2/4 + \frac{9}{2}n^2 \\
 &= 7^iC(n/2^i) + \frac{9}{2}n^2 \sum_{k=0}^{i-1} \left(\frac{7}{4}\right)^k = 7^{\log n}C(1) + \frac{9}{2}n^2 \sum_{k=0}^{\log n-1} \left(\frac{7}{4}\right)^k \\
 &= 7^{\log n} + \frac{9}{2}n^2 \frac{(7/4)^{\log n-1+1} - 1}{7/4 - 1} = 7^{\log n} + \frac{9}{2}n^2 \frac{((7/4)^{\log n} - 1)}{3/4} = \\
 &= n^{\log 7} + 6n^2((7/4)^{\log n} - 1) = n^{\log 7} + 6n^2(n^{\log 7 - \log 4} - 1) = n^{\log 7} + 6n^2\left(\frac{n^{\log 7}}{n^2} - 1\right) \\
 &= O(n^{\log 7}) = O(n^{2.8})
 \end{aligned}$$

136

Quicksort

▣ Thuật toán chia để trị

```
quicksort (A)
begin
  if (n = 1) then return (A)
  else
    Chọn một phần tử x trong danh sách A
    Chia danh sách A thành  $A_1, A_2, A_3$  sao cho các phần tử
    của  $A_1$  nhỏ hơn x, các phần tử của  $A_2$  bằng x và các phần
    tử của  $A_3$  lớn hơn x
    return (quicksort( $A_1$ ),  $A_2$ , quicksort( $A_3$ ))
  endif
end
```

137

Quicksort

▣ Độ phức tạp trong trường hợp trung bình (1)

- $C(n)$: số phép so sánh khi sắp xếp danh sách A có n phần tử khác nhau
- $C(0) = C(1) = 0$
- Giả sử x là phần tử nhỏ thứ i trong danh sách A
 - $|A_1| = i-1$
 - $|A_2| = 1$
 - $|A_3| = n-i$
- Giả sử xác suất phần tử x nhỏ thứ i trong A là $1/n$

138

Quicksort

□ Độ phức tạp trong trường hợp trung bình (2)

- Lời gọi đệ quy cần thời gian trung bình $C(i-1)$ và $C(n-i)$, với $i \in [1..n]$ với xác suất $1/n$
- Để chia A thành A_1 , A_2 và A_3 cần $n-1$ phép so sánh
- Vậy, với $n \geq 2$

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

- Mà

$$\sum_{i=1}^n (C(i-1) + C(n-i)) = \sum_{i=1}^n C(i-1) + \sum_{i=1}^n C(n-i) = \sum_{i=0}^{n-1} C(i) + \sum_{i=0}^{n-1} C(i) = 2 \sum_{i=2}^{n-1} C(i)$$

- Vậy, với $n \geq 2$

$$C(n) = \frac{2}{n} \sum_{i=2}^{n-1} C(i) + n - 1 \quad (1)$$

139

Quicksort

□ Độ phức tạp trong trường hợp trung bình (3)

- Nhân hai vế của (1) với n , với $n \geq 2$ có

$$nC(n) = 2 \sum_{i=2}^{n-1} C(i) + n^2 - n \quad (2)$$

- Thay n bởi $n-1$, với $n \geq 3$ có

$$(n-1)C(n-1) = 2 \sum_{i=2}^{n-2} C(i) + n^2 - 3n + 2 \quad (3)$$

- Lấy (2) trừ (3)

$$nC(n) - (n-1)C(n-1) = 2C(n-1) + 2(n-1)$$

- Vậy

$$nC(n) = (n+1)C(n-1) + 2(n-1) \quad (4)$$

- Chia hai vế của (4) cho $n(n+1)$

$$C(n)/(n+1) = C(n-1)/n + 2(n-1)/n(n+1) \quad (5)$$

140

Quicksort

□ Độ phức tạp trong trường hợp trung bình (4)

- Đặt $S(n) = C(n)/(n+1)$, từ (5) ta có, với $n \geq 3$

$$S(n) = S(n-1) + 2(n-1)/n(n+1) \quad (6)$$

với $S(0) = S(1) = 0$

(5) đúng cả khi $n = 2$, $S(2) = C(2)/3 = 1/3$

- Vậy

$$S(n) \leq \begin{cases} 0 & \text{if } n \leq 1 \\ S(n-1) + 2/n & \text{else} \end{cases}$$

$$S(n) \leq S(n-1) + 2/n \leq S(n-2) + 2/(n-1) + 2/n \\ \leq S(n-3) + 2/(n-2) + 2/(n-1) + 2/n$$

$$\leq S(n-i) + 2 \sum_{k=n-i+1}^n 1/k$$

141

Quicksort

□ Độ phức tạp trong trường hợp trung bình (5)

- Thay $i = n-1$, ta có

$$S(n) \leq S(1) + 2 \sum_{k=2}^n \frac{1}{k} = 2 \sum_{k=2}^n \frac{1}{k} \leq 2 \int_1^n \frac{1}{x} dx = 2 \ln n$$

- Vậy

$$C(n) = (n+1)S(n) \\ \leq 2(n+1) \ln n \\ \leq 2(n+1) \frac{\log n}{\log e} \\ \leq 1.386(n+1) \log n$$

- Vậy $O(n \log(n))$

142

Quicksort

□ Độ phức tạp trong trường hợp xấu nhất

- Danh sách A đã được sắp xếp và ta chọn x là phần tử đầu tiên

$$C(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ C(n-1) + n - 1 & \text{else} \end{cases}$$

- Vậy: $\Theta(n^2)$

143

Quicksort

□ Độ phức tạp trong trường hợp tốt nhất

- Phần tử x được chọn luôn là giá trị trung bình, tức là chia thành hai danh sách con có kích thước $\sim n/2$, nghĩa là $i = n/2$

$$C(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ 2C(n/2) + n - 1 & \text{else} \end{cases}$$

- Vậy: $O(n \log(n))$

144

Quicksort

□ Cài đặt

```
quicksort (A, l, r)
begin
  i = l; j = r
  x = một phần tử thuộc A[l..r]
  repeat
    while (A[i] < x) do i = i + 1 endwhile
    while (x > A[j]) do j = j - 1 endwhile
    if (i ≤ j) then
      hoán đổi A[i] và A[j]
      i = i + 1; j = j - 1
    endif
  until (i > j)
  if (l < j) then quicksort(A, l, j) endif
  if (i < r) then quicksort(A, i, r) endif
end
```

145

Chọn phần tử

□ Bài toán

- Cho danh sách A có n phần tử khác nhau, chọn phần tử lớn thứ k
- Nếu $k = n/2$, vấn đề chọn phần tử trung bình
 - Dùng để chọn chốt trong thuật toán Quicksort

□ Giải pháp đơn giản

- Sắp xếp mảng, sau đó chọn phần tử thứ k
- Độ phức tạp sắp xếp mảng $O(n \log n)$, độ phức tạp bài toán chọn phần tử cũng là $O(n \log n)$

□ Tồn tại giải pháp tốt hơn ?

146

Chọn phần tử

□ Giải pháp tốt hơn (1)

- Ý tưởng
 - chia danh sách A thành các danh sách con, mỗi danh sách có 5 phần tử,
 - tìm phần tử trung bình của các danh sách con,
 - sau đó tìm phần tử trung bình của các phần tử trung bình
- Đề xuất bởi M. R. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan

147

Chọn phần tử

□ Giải pháp tốt hơn (2)

```

F(A, k)
begin
  Chia A thành các danh con có 5 phần tử (danh sách con
  cuối cùng có thể ít hơn 5 phần tử)
   $S_1 = \{\text{các phần tử trung bình từ các danh sách con}\}$ ,  $|S_1|=m$ 
   $x_0 = F(S_1, m/2)$  // phần tử trung bình của các phần tử trung bình
   $S_2 = \{x \in S \mid x < x_0\}$ ,  $S_3 = \{x \in S \mid x > x_0\}$ 
  if ( $|S_2| \leq k$ ) then return (F( $S_2$ , k))
  else if ( $|S_3| \geq n-k+1$ ) then return (F( $S_3$ , n-k+1))
  else return ( $x_0$ )
  endif
endif
end
  
```

148

Chọn phần tử

□ Giải pháp tốt hơn (3)

- Thuật toán cho độ phức tạp trong trường hợp trung bình là $O(n)$
- Chi tiết hơn
 - M. R. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, Time bounds for selection, J. Comput. System Sci. 7 (1972) 448-461

149

Tính bao đóng lồi

□ Bài toán

- Bao đóng lồi của tập hợp E gồm n điểm trong không gian hai chiều là một đa giác lồi sao cho các đỉnh của đa giác là các điểm thuộc E và tất cả các điểm của E đều nằm bên trong hoặc trên các cạnh của đa giác
- Xác định bao đóng lồi của tập E gồm n điểm
- Tính chất
 1. Các đỉnh của đa giác lồi thuộc tập E
 2. Một đường thẳng bất kỳ chia mặt phẳng làm hai phần. Trong mỗi phần, điểm xa đường thẳng nhất trong số n điểm là một đỉnh của đa giác lồi
 3. Một đoạn thẳng nối hai điểm trong số n điểm là một cạnh của đa giác lồi, nếu tất cả các điểm còn lại nằm về một phía của đoạn thẳng

150

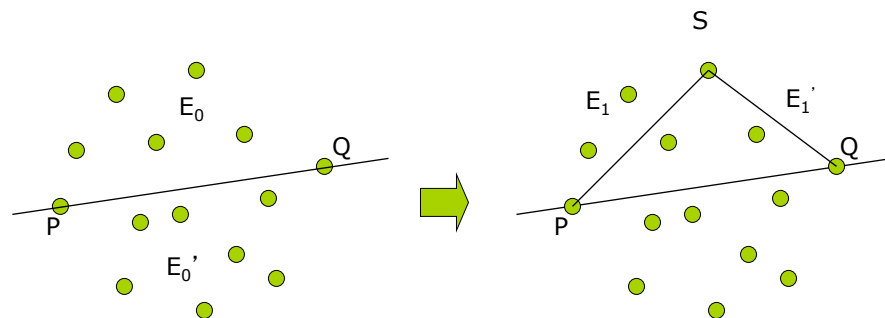
Tính bao đóng lồi

- Tồn tại nhiều thuật toán xác định bao đóng lồi
 - Trong đó có thuật toán chia để trị đề xuất bởi F. Preparata & S.J. Hong
- Ý tưởng
 - Giả sử P và Q là hai điểm của bao đóng lồi (chẳng hạn hai điểm có hoành độ lớn nhất và nhỏ nhất)
 - Đường thẳng PQ chia E thành hai phần E_0 và E_0'
 - Theo tính chất 3, PQ là cạnh của bao lồi E_0 và E_0'
 - Một khi có được bao đóng lồi của E_0 và E_0' thì hợp chúng lại sẽ có được bao đóng lồi của E
 - Xét E_0 , giả sử điểm xa PQ nhất là S , theo tính chất 2, S thuộc bao đóng lồi của E_0
 - Chia E_0 thành E_1 và E_1' :
 - E_1 là phần giới hạn bởi PS không chứa Q
 - E_1' là phần giới hạn bởi QS không chứa P
 - các điểm thuộc tam giác PQS không thuộc bao lồi E_0
 - Tiếp tục áp dụng một cách đệ quy cho E_1 và E_1'

151

Tính bao đóng lồi

□ Minh họa



152

Tính bao đóng lỗi

▣ Thuật toán (1)

```

bao-dong-loi (E)
  begin
    Tính P và Q
    if (hoànch độ P = hoànch độ Q) then
      // tất cả n điểm trên đường thẳng
      return (danh sách các điểm của E)
    else
      Tính  $E_0$  và  $E_0'$ 
      return (hoa-nhap(nua-baoloi( $E_0$ , P, Q), nua-baoloi( $E_0'$ , Q, P)))
    endif
  end

```

153

Tính bao đóng lỗi

▣ Thuật toán (2)

```

nua-baoloi (E, P, Q)
  begin
    Tính S // điểm xa PQ nhất
    if (S trên PQ) then return (PQ) // E rỗng
    else
      Tính  $E_1$  = là phần giới hạn bởi PS không chứa Q
      Tính  $E_1'$  = là phần giới hạn bởi QS không chứa P
      return (hoa-nhap(nua-baoloi( $E_1$ , P, S), nua-baoloi( $E_1'$ , S, Q)))
    endif
  end

```

154

Tính bao đóng lồi

- ▣ Độ phức tạp của thuật toán $O(n \log n)$
- ▣ Chi tiết hơn
 - Franco Preparata & S.J. Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", Comm. ACM 20, 87-93 (1977)

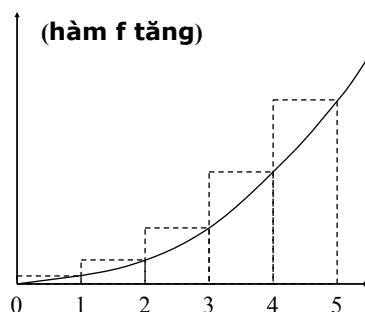
155

Tính tổng

- ▣ Tính tổng (1)

$$F(n) = \sum_{i=1}^n f(i)$$

Ví dụ: $f(x) = x^p$



$$\int_0^5 f(t) dt \leq \sum_{i=1}^5 f(i)$$

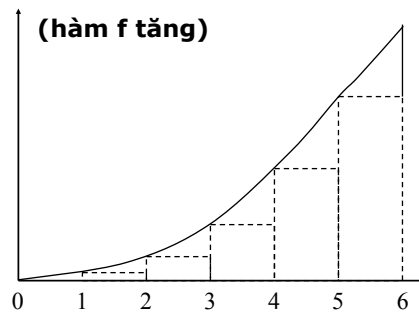
Tổng quát

$$\int_0^n f(t) dt \leq \sum_{i=1}^n f(i)$$

156

Tính tổng

□ Tính tổng (2)



$$\sum_{i=1}^5 f(i) \leq \int_1^6 f(t) dt$$

Tổng quát

$$\sum_{i=1}^n f(i) \leq \int_1^{n+1} f(t) dt$$

vậy:

$$\int_0^n f(t) dt \leq \sum_{i=1}^n f(i) \leq \int_1^{n+1} f(t) dt$$

Nếu hàm f giảm:

$$\int_1^{n+1} f(t) dt \leq \sum_{i=1}^n f(i) \leq \int_0^n f(t) dt$$



157

Bài tập

□ Bài 1

1. Chứng minh rằng có thể nhân hai đa thức $ax+b$ và $cx+d$ chỉ với 3 phép nhân (gợi ý: một trong những phép nhân $(a+b)(c+d)$)
2. Xây dựng hai thuật toán chia để trị nhân hai đa thức với độ phức tạp $O(n^{\log_2 3})$
 - a. Thuật toán thứ nhất cần phải chia đôi đa thức thành hai đa thức, một nửa có bậc $n/2$ (mũ $[0..n/2]$) và một nửa có bậc n (mũ $[n/2+1..n]$)
 - b. Thuật toán thứ hai cần phải chia đôi đa thức thành hai đa thức, một nửa có mũ là chẵn, một nửa có mũ là lẻ
3. Chứng minh rằng hai số nguyên được biểu diễn bởi n bit có thể được nhân bởi thuật toán có độ phức tạp $O(n^{\log_2 3})$

158

Quy hoạch động (6)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường Đại học Bách khoa
Đại học Đà Nẵng

Quy hoạch động (dynamic programming)

- Nguyên tắc tương tự thuật toán chia để trị
 - Bài toán được chia thành nhiều bài toán con
 - Bài toán tiếp tục được chia thành các bài toán con khác, cho đến khi các bài toán con có thể giải quyết được dễ dàng
 - Kết hợp giải pháp của các bài toán con có được giải pháp của bài toán ban đầu

160

Quy hoạch động

- Sự khác nhau với thuật toán chia để trị
 - Quy hoạch động được áp dụng khi các bài toán con không độc lập
 - Các bài toán con chung
 - Khi các bài toán con không độc lập
 - Áp dụng thuật toán chia để trị
 - Thực hiện cùng công việc (giải quyết cùng một bài toán con) nhiều lần
 - Áp dụng thuật toán quy hoạch động
 - Mỗi bài toán con được giải quyết một lần và ghi kết quả vào một mảng, sau đó nếu gặp lại bài toán con đó chỉ lấy kết quả sử dụng
 - Giảm độ phức tạp

161

Quy hoạch động

- Quy hoạch động = Chia để trị + bộ nhớ phụ
- Chia để trị : tiếp cận từ trên xuống
 - Giải quyết bài toán lớn trước sau đó giải quyết bài toán con sau
- Quy hoạch động : tiếp cận từ dưới lên
 - Giải quyết bài toán con trước sau đó dựa trên các bài toán con đã giải quyết, giải quyết bài toán lớn sau

162

Quy hoạch động

- ▣ Thuật toán quy hoạch động thường được áp dụng cho các **bài toán tối ưu**
 - Các bài toán này có thể có nhiều giải pháp, chúng ta muốn tìm giải pháp tối ưu theo một hàm mục tiêu
- ▣ Xây dựng thuật toán quy hoạch động thường trải qua các bước
 - Xác định các tính chất của cấu trúc của giải pháp tối ưu
 - Định nghĩa đệ quy giá trị của giải pháp tối ưu
 - Tính giá trị của giải pháp tối ưu thông qua các trường hợp đơn giản (trường hợp dừng của đệ quy) và lần lên cho đến khi giải quyết được bài toán ban đầu
 - Xây dựng giải pháp tối ưu đối với các thông tin vừa tính toán
 - ▣ Nếu cần cả giải pháp tối ưu chứ không chỉ là giá trị của giải pháp tối ưu

163

Một số ứng dụng

- ▣ Triển khai nhị thức $(a+b)^n$
- ▣ Nhân dãy ma trận
- ▣ Dãy con chung dài nhất
- ▣ Xếp ba lô

164

Triển khai nhị thức $(a+b)^n$

- ▣ Nhị thức $(a+b)^n$ được triển khai theo công thức sau

$$(a+b)^n = \sum_{k=0}^n C_n^k a^{n-k} b^k$$

- ▣ Với

$$C_n^k = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n!}{k!(n-k)!}$$

- ▣ Công thức cho phép tính tổ hợp chập k của n

$$C_n^k = \begin{cases} 1 & k=0, k=n \\ C_{n-1}^{k-1} + C_{n-1}^k & 1 \leq k \leq n-1 \end{cases}$$

165

Triển khai nhị thức $(a+b)^n$

- ▣ Thuật toán chia để trị tính C_n^k (1)

```

C(k, n)
begin
  if (k = 0 or k = n) then
    return (1)
  else
    return (C(k-1, n-1) + C(k, n-1))
end

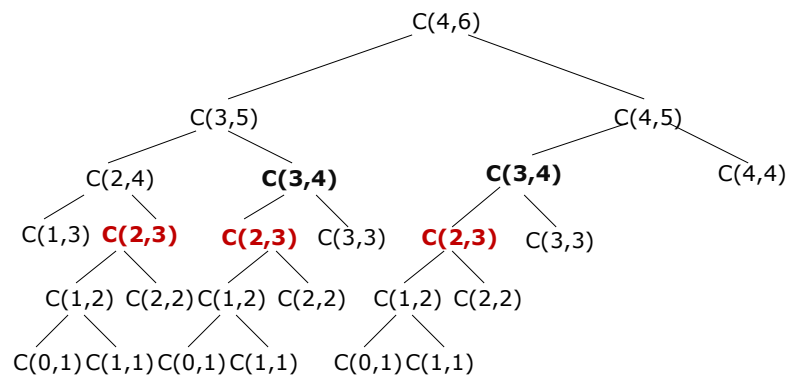
```

- Có nhiều giá trị $C(i, j)$ với $i < k$ và $j < n$, được tính lặp nhiều lần
- Độ phức tạp lớn

166

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán chia để trị tính C_n^k (2)



167

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán chia để trị tính C_n^k (3)

■ Tính độ phức tạp

- ▣ Gọi $C(n)$ là thời gian thực thi trong trường hợp xấu nhất tính $C(k, n)$ với mọi k
- ▣ Vậy từ thuật toán, ta có

$$C(n) = \begin{cases} c & \text{if } n = 1 \\ 2C(n-1) + d & \text{else} \end{cases}$$

với c và d là các hằng số

168

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán chia để trị tính C_n^k (4)

■ Tính độ phức tạp

$$\begin{aligned}
 C(n) &= 2C(n-1) + d \\
 &= 2(C(n-2) + d) + d = 4C(n-2) + 2d + d \\
 &= 4(2C(n-3) + d) + 2d + d = 8C(n-3) + 4d + 2d + d \\
 &= 2^i C(n-i) + d \sum_{k=0}^{i-1} 2^k \\
 &= 2^{n-1} C(1) + d \sum_{k=0}^{n-2} 2^k = 2^{n-1} c + d \frac{2^{n-1} - 1}{2 - 1} \\
 &= 2^{n-1} (c + d) - d
 \end{aligned}$$

■ Vậy: $C(n) = \Theta(2^n)$

169

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (1)

■ Sử dụng mảng $C[0..n, 0..k]$ lưu các kết quả trung gian

- ▣ $C[i, j]$ chứa giá trị C_i^j
- ▣ $C[i, j]$ được tính

$$C[i, j] = \begin{cases} 1 & j = 0 \text{ hay } j = i \\ C[i-1, j-1] + C[i-1, j] & \text{trường hợp khác} \end{cases}$$

▣ Tính các giá trị của tam giác Pascal

n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

170

Triển khai nhị thức $(a+b)^n$

- ▣ Thuật toán quy hoạch động tính C_n^k (2)

```

C1(k, n)
begin
  // Khởi gán cột 0 và đường chéo
  for i from 0 to n-k do C[i,0] = 1 endfor
  for i from 0 to k do C[i,i] = 1 endfor
  // Tính từng cột
  for j from 1 to k do
    for i from j+1 to n-k+j do
      C[i,j] = C[i-1,j-1] + C[i-1,j]
    endfor
  endfor
  return (C[n,k])
end

```

171

Triển khai nhị thức $(a+b)^n$

- ▣ Thuật toán quy hoạch động tính C_n^k (3)

- Ví dụ: $n=8, k=5$

n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1		1			
3	1			1		
4					1	
5						1
6						
7						
8						

→

n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4		4	6	4	1	
5			10	10	5	1
6				20	15	6
7					35	21
8						56

Khởi gán cột 0 và đường chéo

Tính theo từng cột

Chỉ cần tính các giá trị cần cho việc tính $C[8,5]$

172

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (4)

- Độ phức tạp
 - ▣ Số các giá trị cần tính (giả sử phép cộng là phép toán cơ bản)

$$k(n-k) = nk - k^2 \leq nk$$
 - ▣ Vậy độ phức tạp về thời gian $O(nk)$
 - ▣ Sử dụng mảng có nk ô nhớ để lưu trữ các giá trị trung gian, hay độ phức tạp về mặt không gian $O(nk)$
- Có thể cải tiến thuật toán để giảm số ô nhớ sử dụng ?

173

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (5)

Chỉ sử dụng mảng
một chiều lưu trữ
dòng hiện thời của
tam giác Pascal

```

C2(k, n)
begin
  // sử dụng mảng một chiều C[0..n]
  C[0] = 1 // khởi gán hàng 1
  C[1] = 1
  // tính từng hàng
  for i from 2 to n do
    p1 = 1
    for j from 1 to i-1 do
      p2 = C[j]
      C[j] = p1 + p2
      p1 = p2
    endfor
    C[i] = 1 // phần tử trên đường chéo
  endfor
  return (C[k])
end
  
```

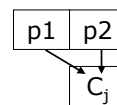
174

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (6)

■ Ví dụ: $n=8, k=5$

n/k	0	1	2	3	4	5	6	7	8
0									
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1



175

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (7)

■ Độ phức tạp

▣ Số các giá trị cần tính (giả sử phép cộng là phép toán cơ bản)

$$\sum_{i=2}^n (i-1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$

▣ Vậy độ phức tạp về thời gian $O(n^2)$

▣ Sử dụng mảng có n ô nhớ để lưu trữ các giá trị trung gian, hay độ phức tạp về mặt không gian $O(n)$

176

Thiết kế thuật toán quy hoạch động

□ Nhận dạng

- Xây dựng thuật toán chia để trị/thuật toán đơn giản
- Đánh giá độ phức tạp (hàm mũ)
- Cùng một bài toán con được giải quyết nhiều lần

□ Xây dựng

- Tách phần « trị » trong thuật toán chia để trị và thay thế các lời gọi đệ quy bằng việc tìm kiếm các giá trị trong một mảng
- Thay vì trả về giá trị, ghi giá trị vào mảng
- Sử dụng điều kiện dừng của thuật toán chia để trị để khởi tạo giá trị của mảng
- Tìm cách tính các giá trị của mảng
- Xây dựng vòng lặp để tính các giá trị của mảng

177

Thiết kế thuật toán quy hoạch động

```

C(k, n)
begin
  if (k = 0 or k = n) then return (1)
  else return (C(k-1, n-1) + C(k, n-1))
end

```

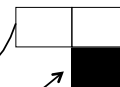
Chia để trị

```

C1(k, n)
begin
  for i from 0 to n-k do C[i,0] = 1 endfor
  for i from 0 to k do C[i,i] = 1 endfor
  for j from 1 to k do
    for i from j+1 to n-k+j do
      C[i,j] = C[i-1,j-1] + C[i-1,j]
    endfor
  endfor
  return (C[n,k])
end

```

Quy hoạch động



178

Nhân dãy ma trận

□ Bài toán

- Có n ma trận M_1, M_2, \dots, M_n , cần tính tích $M_1.M_2.\dots.M_n$ sao cho thực hiện ít phép nhân nhất
- Phép nhân ma trận có tính kết hợp
 - Có thể thực hiện tích $M_1.M_2.\dots.M_n$ bởi nhiều thứ tự kết hợp khác nhau
- Giả thiết sử dụng thuật toán đơn giản để nhân hai ma trận
 - Nhân hai ma trận có kích thước $p \times q$ và $q \times r$ cần thực hiện $p \times q \times r$ phép nhân

179

Nhân dãy ma trận

□ Ví dụ

- Nhân dãy ma trận
 $M_1(10 \times 20).M_2(20 \times 50).M_3(50 \times 1).M_4(1 \times 100)$
- Có các 5 cách kết hợp
 - $(M_1.(M_2.(M_3.M_4)))$:
 $50 \times 1 \times 100 + 20 \times 50 \times 100 + 10 \times 20 \times 100 = 125000$ phép nhân
 - $(M_1.((M_2.M_3).M_4))$: 72000
 - $((M_1.M_2).(M_3.M_4))$: 65000
 - $((M_1.(M_2.M_3)).M_4)$: 2200
 - $((((M_1.M_2).M_3).M_4))$: 60500
- Vấn đề: cách kết hợp nào thực hiện ít phép nhân nhất?

180

Nhân dãy ma trận

▣ Thuật toán vét cạn (1)

- Thử tất cả các cách kết hợp có thể
- Tính số phép nhân cho mỗi cách
- Chọn cách tốt nhất

- Số tất cả các cách kết hợp là **hàm mũ**

- Gọi $P(n)$ là số cách kết hợp n ma trận
 - ▣ Khi $n = 1$ thì $P(1) = 1$
 - ▣ Khi $n \geq 2$ thì có thể chia tích $M_1 \cdot M_2 \dots M_n$ thành hai

$$M_1 \cdot M_2 \dots M_n = \underbrace{(M_1 \cdot M_2 \dots M_k)}_{P(k) \text{ cách}} \cdot \underbrace{(M_{k+1} \dots M_n)}_{P(n-k) \text{ cách}}$$

có $n-1$ cách chia ($k=1..n-1$)

181

Nhân dãy ma trận

▣ Thuật toán vét cạn (2)

- Vậy

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n \geq 2 \end{cases}$$

- Có thể chỉ ra được

$$P(n) = \Omega(2^n) \quad \text{!}$$

182

Nhân dãy ma trận

□ Thuật toán chia để trị (1)

- Gọi $m_{i,j}$ là **số tối thiểu** các phép nhân khi thực hiện tích $M_i M_{i+1} \dots M_j$
 - Rõ ràng $m_{i,i} = 0$
- Giả sử kích thước các ma trận M_i, M_{i+1}, \dots, M_j lần lượt là $(d_{i-1}, d_i), (d_i, d_{i+1}), \dots, (d_{j-1}, d_j)$
- Giả sử rằng, chúng ta biết được cách kết hợp $M_i M_{i+1} \dots M_j$ với chi phí tối thiểu ($m_{i,j}$ nhỏ nhất) là $(M_i M_{i+1} \dots M_k) \cdot (M_{k+1} \dots M_j)$
 - nghĩa là k đã biết
 - khi đó:

$$m_{i,j} = m_{i,k} + m_{k+1,j} + \text{số phép nhân để nhân hai ma trận } (d_{i-1}, d_k) \cdot (d_k, d_j)$$

$$m_{i,j} = m_{i,k} + m_{k+1,j} + d_{i-1} \cdot d_k \cdot d_j$$
- Tuy nhiên, k chưa xác định, $k \in [1..n-1]$, vậy:

$$m_{i,j} = \begin{cases} \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1} d_k d_j) & i < j \end{cases}$$

183

Nhân dãy ma trận

□ Thuật toán chia để trị (2)

- Từ đó xây dựng thuật toán chia để trị cho phép tính $m_{i,j}$ với tất cả các giá trị có thể của k

```

matrixChainRecur(i, j)
m = +∞
begin
  if (i=j) then m = 0
  else for k from i to j-1 do
    r = matrixChainRecur(i,k) + matrixChainRecur(k+1,j) + di-1dkdj
    if (r < m) then
      m = r
    endif
  endfor
endif
return (m)
end

```

Nghiệm của bài toán là $\text{matrixChainRecur}(1,n)$

184

Nhân dãy ma trận

Thuật toán quy hoạch động (1)

- Chúng ta cần xây dựng mảng $m[1..n, 1..n]$
- Với mỗi $m[i, j]$, $i \leq j$, nên chỉ nửa trên đường chéo chính của mảng m được sử dụng
- Mỗi $m[i, j]$ được tính theo công thức

$$m_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1}d_kd_j) & i < j \end{cases}$$

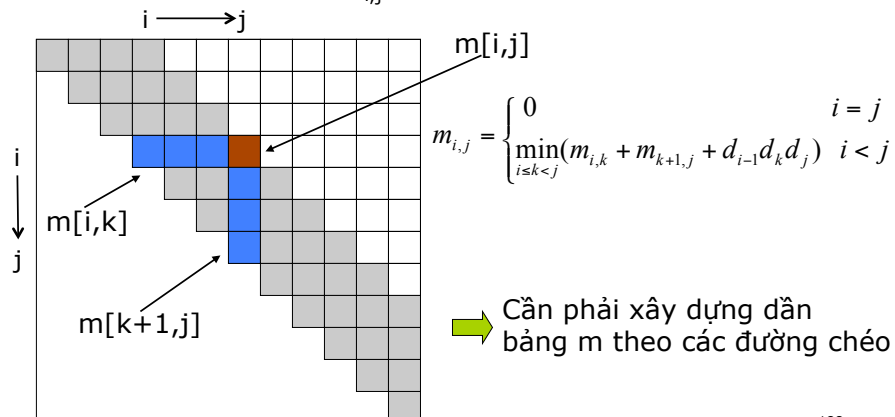
- Vậy, muốn tính $m_{i,j}$ chúng ta phải có giá trị $m_{i,k}$ và $m_{k+1,j}$ với $i \leq k < j$

187

Nhân dãy ma trận

Thuật toán quy hoạch động (2)

- Minh họa cách tính $m_{i,j}$



188

Nhân dãy ma trận

Thuật toán quy hoạch động (3)

- Mảng m sẽ được xây dựng dần theo từng đường chéo
- Đường chéo s sẽ gồm các phần tử m[i,j] mà j-i=s
- Vậy

$$m_{i,j} = \begin{cases} 0 & s = 0 \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1}d_kd_{i+s}) & 0 < s < n \end{cases}$$

- Xây dựng mảng m mới chỉ cho phép tính số phép nhân tối thiểu (giá trị) của giải pháp
- Cần phải xây dựng giải pháp (cách thực hiện các phép nhân ma trận)
 - Sử dụng mảng phụ g[1..n,1..n] để ghi nhớ chỉ số k cho giải pháp tối ưu mỗi bước

189

Nhân dãy ma trận

Thuật toán quy hoạch động (4)

```

matrixChain(n)
begin
  for i from 0 to n do m[i,i] = 0 endfor
  for s from 1 to n-1 do // tính tất cả các đường chéo s
    for i from 1 to n-s do // tính một đường chéo s
      j = i + s
      // tính mi,j = min(mi,k + mk+1,j + di-1dkdi+s)
      m[i,j] = +∞
      for k from i to j-1 do
        r = m[i,k] + m[k+1,j] + di-1dkdj
        if (r < m[i,j]) then
          m[i,j] = r
          g[i,j] = k // ghi nhớ k
        endif
      endfor
    endfor
  endfor
  return m và g
end
  
```

190

Nhân dãy ma trận

▣ Thuật toán quy hoạch động (5)

■ Ví dụ

▣ $M_1(10 \times 20) \cdot M_2(20 \times 50) \cdot M_3(50 \times 1) \cdot M_4(1 \times 100)$
nghĩa là: $d_0=10, d_1=20, d_2=50, d_3=1, d_4=100$

▣ Xây dựng đường chéo $s = 1$

- ▣ $m[1,2] = \min(m[1,k] + m[k+1,2] + d_0 d_1 d_2)$, với $1 \leq k < 2$
 $= \min(m[1,1] + m[2,2] + d_0 d_1 d_2)$
 $= d_0 d_1 d_2 = 10 \times 20 \times 50 = 10000$
 $g[1,2] = k = 1$
- ▣ $m[2,3] = \min(m[2,k] + m[k+1,3] + d_1 d_2 d_3)$, với $2 \leq k < 3$
 $= d_1 d_2 d_3 = 20 \times 50 \times 1 = 1000$
 $g[2,3] = k = 2$
- ▣ $m[3,4] = d_2 d_3 d_4 = 50 \times 1 \times 100 = 5000$
 $g[3,4] = k = 3$

191

Nhân dãy ma trận

▣ Thuật toán quy hoạch động (6)

■ Ví dụ

▣ Xây dựng đường chéo $s = 2$

- ▣ $m[1,3] = \min(m[1,k] + m[k+1,3] + d_0 d_k d_3)$, với $1 \leq k < 3$
 $= \min(m[1,1] + m[2,3] + d_0 d_1 d_3,$
 $\quad m[1,2] + m[3,3] + d_0 d_2 d_3)$
 $= \min(0 + 1000 + 200, 10000 + 0 + 500)$
 $= 1200$
 $g[1,3] = k = 1$
- ▣ $m[2,4] = \min(m[2,k] + m[k+1,4] + d_1 d_k d_4)$, với $2 \leq k < 4$
 $= \min(m[2,2] + m[3,4] + d_1 d_2 d_4,$
 $\quad m[2,3] + m[4,4] + d_1 d_3 d_4)$
 $= \min(0 + 5000 + 100000, 1000 + 0 + 2000)$
 $= 3000$
 $g[2,4] = k = 3$

192

Nhân dãy ma trận

Thuật toán quy hoạch động (7)

■ Ví dụ

□ Xây dựng đường chéo $s = 3$

$$\begin{aligned}
 m[1,4] &= \min(m[1,k] + m[k+1,4] + d_0 d_k d_4), \text{ với } 1 \leq k < 4 \\
 &= \min(m[1,1] + m[2,4] + d_0 d_1 d_4, \\
 &\quad m[1,2] + m[3,4] + d_0 d_2 d_4, \\
 &\quad m[1,3] + m[4,4] + d_0 d_3 d_4) \\
 &= \min(0 + 3000 + 20000, \\
 &\quad 10000 + 5000 + 50000, \\
 &\quad 1200 + 0 + 1000) \\
 &= 2200 \\
 g[1,4] &= k = 3
 \end{aligned}$$

□ $m[1,4]$ chính là kết quả cần tìm

193

Nhân dãy ma trận

Thuật toán quy hoạch động (8)

■ Ví dụ: các giá trị mảng m và g

0	10000 1	1200 1	2200 3	s=3
	0	1000 2	3000 3	s=2
		0	5000 3	s=1
			0	s=0

194

Nhân dãy ma trận

□ Thuật toán quy hoạch động (9)

- Đánh giá độ phức tạp
 - Thuật toán gồm 3 vòng lặp lồng nhau
 - Mỗi vòng lặp đều không lặp quá n lần
 - Vậy độ phức tạp $O(n^3)$
- Thuật toán quy hoạch động tốt hơn so với thuật toán đơn giản và thuật toán chia để trị

195

Nhân dãy ma trận

□ Thuật toán quy hoạch động (10)

- Xây dựng giải pháp tối ưu
 - thực hiện các phép nhân theo thứ tự tối ưu
 - *chainmatrix* chỉ tính giá trị tối ưu của cách kết hợp các phép nhân, chứ không thực hiện phép nhân
 - Sử dụng thông tin chứa trong mảng g
 - $g[i,j]$ chứa giá trị k , mà tích $M_i M_{i+1} \dots M_j$ được tách đôi giữa M_k và M_{k+1}
- Giả sử đã có thuật toán nhân hai ma trận X và Y :
`matrixProduct(X, Y)`

196

Nhân dãy ma trận

□ Thuật toán quy hoạch động (11)

- Xây dựng giải pháp tối ưu

```
chainMatrixProduct(M, g, i, j)
// dãy các ma trận
begin
  if (i < j) then
    X = chainMatrixProduct(M, g, i, g[i,j])
    Y = chainMatrixProduct(M, g, g[i,j]+1,j)
    return (matrixProduct(X,Y))
  else // i = j
    return (Mi)
  endif
end
```

- Khi gọi chainMatrixProduct(M, g, 1, 4) sẽ tính tích $M_1.M_2.M_3.M_4$ theo thứ tự: $((M_1.(M_2.M_3)).M_4)$
- Vì: $g[1,4] = 3, g[1,3] = 1$

197

Nhân dãy ma trận

□ Thuật toán quy hoạch động (12)

- Bài tập

- Viết thuật toán in ra biểu thức kết hợp tối ưu thực hiện nhân dãy ma trận

- Ví dụ

Cho dãy ma trận: $M_1(10 \times 20).M_2(20 \times 50).M_3(50 \times 1).M_4(1 \times 100)$

Kết quả nhận được: $((M_1.(M_2.M_3)).M_4)$

198

Dãy con chung dài nhất

□ Bài toán

- Cho hai dãy kí hiệu X và Y, dãy con chung dài nhất (Longest Common Subsequence - LCS) của X và Y là dãy các kí hiệu nhận được từ X bằng cách xoá đi một số các phần tử và cũng nhận được từ Y bằng cách xoá đi một số phần tử
- Ví dụ: X = ABCBDAB và Y = BDCABA

X = A **B** **C** **B** D **A** B
 Y = **B** D **C** A **B** **A**
 Dãy con chung dài nhất: **BCBA**
- Ứng dụng so sánh « độ tương tự » hai chuỗi ADN

199

Dãy con chung dài nhất

□ Thuật toán vét cạn

- Giả sử $X = x_1x_2...x_n$ và $Y = y_1y_2...y_m$
- So sánh mỗi dãy kí hiệu con của X với dãy kí hiệu Y
 - Có 2^n dãy con của X
 - Mỗi dãy con của X so sánh với Y sẽ thực hiện m phép so sánh
- Độ phức tạp sẽ là $O(m2^n)$
 - Hàm mũ !

200

Dãy con chung dài nhất

Thuật toán chia để trị (1)

- Có thể phân tích bài toán thành những bài toán con với kích thước nhỏ hơn
 - Bài toán con: tìm dãy con chung dài nhất của các cặp *tiền tố* của X và Y
 - Cho $X = x_1x_2...x_n$, $X_i = x_1x_2...x_i$ được gọi là tiền tố thứ i của X
- Gọi $Z = z_1z_2...z_k$ là dãy con chung dài nhất (LCS) của $X = x_1x_2...x_n$ và $Y = y_1y_2...y_m$
 - Nếu $x_n = y_m$ thì $z_k = x_n = y_m$ và $Z_{k-1} = \text{LCS}(X_{n-1}, Y_{m-1})$
 - Nếu $x_n \neq y_m$ và $z_k \neq x_n$ thì $Z = \text{LCS}(X_{n-1}, Y)$
 - Nếu $x_n \neq y_m$ và $z_k \neq y_m$ thì $Z = \text{LCS}(X, Y_{m-1})$

201

Dãy con chung dài nhất

Thuật toán chia để trị (2)

- Giải pháp đệ quy
 - Gọi $c[i, j]$ là độ dài dãy con chung dài nhất của X_i và Y_j
 - Khi đó, độ dài dãy con chung dài nhất của X và Y sẽ là $c[n, m]$
 - Độ dài dãy con chung dài nhất của một dãy rỗng và một dãy bất kỳ luôn bằng 0
 - $c[i, 0] = 0$ và $c[0, j] = 0$ với mọi i, j
 - Vậy, ta có

$$c[i, j] = \begin{cases} 0 & i = 0 \text{ hay } j = 0 \\ c[i-1, j-1] + 1 & i, j > 0, x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & i, j > 0, x_i \neq y_j \end{cases}$$

202

Dãy con chung dài nhất

▣ Thuật toán chia để trị (3)

```

LCS-length (i,j)
begin
  if (i = 0 or j = 0) then
    return (0)
  else
    if (xi = yj) then
      return (LCS-length(i-1,j-1) + 1)
    else
      return (max(LCS-length(i-1, j), LCS-length(i, j-1)))
    endif
  endif
end

```

203

Dãy con chung dài nhất

▣ Thuật toán chia để trị (4)

- Đánh giá độ phức tạp
 - ▣ Giả sử $n \geq m$
 - ▣ Gọi $C(k)$ là thời gian thực thi trong **trường hợp xấu nhất**
 - ▣ Vậy: $C(k) \geq 2C(m-1) = 2^2C(m-2) = 2^mC(0) = 2^m$
 - ▣ Nghĩa là thuật toán có độ phức tạp $\Omega(2^m)$

204

Dãy con chung dài nhất

Thuật toán quy hoạch động (1)

- Lưu giá trị độ dài của các dãy con chung dài nhất các cặp tiền tố vào mảng $c[0..n, 0..m]$

```

LCS-length (X,Y)
begin
  n = length(X)
  m = length(Y)
  for i from 0 to n do c[i,0]=0 endfor
  for j from 0 to m do c[0,j]=0 endfor
  for i from 1 to n do
    for j from 1 to m do
      if (xi = yj) then c[i,j] = c[i-1,j-1] + 1
      else
        c[i,j] = max(c[i-1,j], c[i,j-1])
      endif
    endfor
  endfor
  return c
end
  
```

205

Dãy con chung dài nhất

Thuật toán quy hoạch động (2)

- Ví dụ

		j	0	1	2	3	4	5
		i	y _j	B	D	C	A	B
0	x _i							
1	A							
2	B							
3	C							
4	B							

X=ABCB, Y=BDCAB

206

Dãy con chung dài nhất

Thuật toán quy hoạch động (3)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0				
2	B	0				
3	C	0				
4	B	0				

for i from 0 to n do c[i,0]=0
for j from 0 to m do c[0,j]=0

207

Dãy con chung dài nhất

Thuật toán quy hoạch động (4)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0			
2	B	0				
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

208

Dãy con chung dài nhất

Thuật toán quy hoạch động (5)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	
2	B	0				
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

209

Dãy con chung dài nhất

Thuật toán quy hoạch động (6)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	1
2	B	0				
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

210

Dãy con chung dài nhất

Thuật toán quy hoạch động (7)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	1 → 1
2	B	0				
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

211

Dãy con chung dài nhất

Thuật toán quy hoạch động (7)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	1
2	B	0	1			
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

212

Dãy con chung dài nhất

Thuật toán quy hoạch động (8)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	1
2	B	0	1	1	1	
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

213

Dãy con chung dài nhất

Thuật toán quy hoạch động (9)

■ Ví dụ

j	0	1	2	3	4	5
i	y_j	B	D	C	A	B
0	x_i	0	0	0	0	0
1	A	0	0	0	0	1
2	B	0	1	1	1	2
3	C	0				
4	B	0				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

214

Dãy con chung dài nhất

Thuật toán quy hoạch động (10)

■ Ví dụ

du

j	0	1	2	3	4	5	
i	y_j	B	D	C	A	B	
0	x_i	0	0	0	0	0	
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1			
4	B	0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

215

Dãy con chung dài nhất

Thuật toán quy hoạch động (11)

■ Ví dụ

	j	0	1	2	3	4	5
i		y_j	B	D	C	A	B
0	x_i	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2		
4	B	0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

216

Dãy con chung dài nhất

Thuật toán quy hoạch động (12)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y _j	B	D	C	A	B
0	x _i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0					

if (x_i = y_j) then c[i,j] = c[i-1,j-1] + 1
 else c[i,j] = max(c[i-1,j], c[i,j-1])

217

Dãy con chung dài nhất

Thuật toán quy hoạch động (13)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y _j	B	D	C	A	B
0	x _i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1				

if (x_i = y_j) then c[i,j] = c[i-1,j-1] + 1
 else c[i,j] = max(c[i-1,j], c[i,j-1])

218

Dãy con chung dài nhất

Thuật toán quy hoạch động (14)

■ Ví dụ

	j	0	1	2	3	4	5
i	y_j	B	D	C	A	B	
0	x_i	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	1	1	2	2	

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

219

Dãy con chung dài nhất

Thuật toán quy hoạch động (15)

■ Ví dụ

du

j

0

1

2

3

4

5

i

y_j

B

D

C

A

B

0

x_i

0

0

0

0

0

0

1

A

0

0

0

0

1

1

2

B

0

1

1

1

1

2

3

C

0

1

1

2

2

2

4

B

0

1

1

2

2

3

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

220

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (16)

- Đánh giá độ phức tạp
 - Thuật toán nhằm tính mảng c có $n \times m$ phần tử
 - Tính mỗi phần tử cần $O(1)$ thời gian
 - Vậy độ phức tạp của thuật toán là $O(nm)$

221

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (17)

- Xây dựng giải pháp tối ưu
 - Thuật toán *LCS-length* chỉ tính độ dài của dãy con chung dài nhất
 - Cần xác định dãy con chung dài nhất đó
 - Tương tự như thuật toán nhân dãy ma trận, sử dụng mảng phụ $b[1..n, 1..m]$ ghi nhớ các phần tử của mảng c ứng với giải pháp tối ưu
 - Khi $c[i, j] = c[i-1, j-1] + 1$ thì ghi nhớ x_i , vì x_i thuộc dãy con chung dài nhất
 - Xuất phát từ $c[m, n]$ lần ngược lên
 - Nếu $c[i, j] = c[i-1, j-1] + 1$ thì x_i thuộc dãy con chung dài nhất
 - Nếu $i = 0$ hoặc $j = 0$ thì dừng lại
 - Đảo ngược dãy kí hiệu nhận được dãy con chung dài nhất

222

Dãy con chung dài nhất

Thuật toán quy hoạch động (18)

■ Ví dụ

j	0	1	2	3	4	5
Y _j		B	D	C	A	B
i						
0	X _i	0	0	0	0	0
1	A	0	0	0	1	1
2	B	0	1	1	1	2
3	C	0	1	1	2	2
4	B	0	1	1	2	3

Dãy con chung dài nhất (thứ tự ngược) **B C B**
 Dãy con chung dài nhất **B C B**

223

Dãy con chung dài nhất

Thuật toán quy hoạch động (20)

- Chỉnh lại thuật toán *LCS-length*, ghi nhớ giải pháp tối ưu vào mảng b

```

LCS-length (X,Y)
begin
  n = length(X), m = length(Y)
  for i from 0 to n do c[i,0]=0 endfor
  for j from 0 to m do c[0,j]=0 endfor
  for i from 1 to n do
    for j from 1 to m do
      if (xi = yj) then c[i,j] = c[i-1,j-1] + 1
                        b[i,j] = '\
      else
        if (c[i-1,j] < c[i,j-1]) then c[i,j] = c[i,j-1]
                                      b[i,j] = '←'
        else c[i,j] = c[i-1,j]
              b[i,j] = '↑'
        endif endif endfor endfor
  return c
end
  
```

224

Dãy con chung dài nhất

Thuật toán quy hoạch động (19)

Xây dựng giải pháp tối ưu

```

print-LCS(b, X, i, j)
begin
  if (i = 0 or j = 0) then
    return
  else
    if (b[i,j] = '\') then // xi thuộc dãy con chung dài nhất
      print-LCS(b,X,i-1,j-1)
      print xi
    else
      if (b[i,j]='←') then
        print-LCS(b,X,i,j-1)
      else // (b[i,j]='↑')
        print-LCS(b,X,i-1,j)
      endif
    endif
  endif
end
  
```






225

Xếp ba lô

Bài toán

- Có một ba lô có thể chứa tối đa trọng lượng W và có n đồ vật, mỗi đồ vật có trọng lượng w_i và giá trị b_i
 - W , w_i và b_i là các số nguyên
- Hãy xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất

Minh hoạ

	Đồ vật	Trọng lượng - w_i	Giá trị - b_i
<div style="border: 1px solid black; padding: 5px; display: inline-block;"> $W = 12$ </div> Ba lô		2	8
		5	10
		3	6
		9	2
		4	3

226

Xếp ba lô

▣ Thuật toán vét cạn

- Có n đồ vật, nên có 2^n tập con các đồ vật của tập n đồ vật
- Duyệt tất cả 2^n tập con các đồ vật và chọn tập có tổng giá trị lớn nhất mà tổng trọng lượng nhỏ hơn W
- Độ phức tạp thuật toán là $O(2^n)$
 - ▣ Hàm mũ !

227

Xếp ba lô

▣ Thuật toán chia để trị (1)

- Bài toán có thể chia thành các bài toán con
 - ▣ Thay vì xét k đồ vật, xét $k-1$ đồ vật ...
- Gọi $v_{k,w}$ là tổng giá trị lớn nhất của ba lô mà trọng lượng không vượt quá w khi chỉ sử dụng các đồ vật $1...k$
- Đối với mỗi đồ vật k cần trả lời câu hỏi
 - ▣ Ba lô hiện tại có thể chứa thêm đồ vật k hay không ?
- Trả lời
 - ▣ Nếu trọng lượng còn lại hiện tại w của ba lô nhỏ hơn w_k thì ba lô không thể chứa đồ vật k
 - ▣ Ngược lại, $w \geq w_k$ thì có hai trường hợp xảy ra:
 - Không thêm đồ vật k vào ba lô, chỉ sử dụng các đồ vật $1...k-1$
 - Thêm đồ vật k vào ba lô, khi đó giá trị của ba lô tăng lên b_k nhưng trọng lượng còn lại của ba lô giảm đi w_k (tức là bằng $w - w_k$)
 - Trường hợp nào cho tổng giá trị của ba lô lớn nhất sẽ được chọn

228

Xếp ba lô

▣ Thuật toán chia để trị (2)

- Vậy, $v_{k,w}$ được tính như sau

$$v_{k,w} = \begin{cases} v_{k-1,w} & \text{if } w_k > w \\ \max \{v_{k-1,w}, v_{k-1,w-w_k} + b_k\} & \text{else} \end{cases}$$

- Giải thích

- ▣ Nếu $w_k > w$, không thể thêm đồ vật k vào ba lô, chỉ xét các đồ vật 1...k-1 (gồm k-1 đồ vật)
- ▣ Nếu $w_k \leq w$,
 - Nếu chỉ cần sử dụng các đồ vật 1...k-1 mà cho tổng giá trị ba lô lớn nhất, thì không sử dụng đồ vật k
 - Sử dụng đồ vật k nếu cho tổng giá trị ba lô lớn nhất, khi đó xét các đồ vật 1...k-1 với trọng lượng còn lại của ba lô là $w - w_k$

229

Xếp ba lô

▣ Thuật toán chia để trị (3)

- Lưu ý rằng, $v_{k,w} = 0$ nếu $k = 0$
- Vậy công thức đầy đủ tính $v_{k,w}$ là

$$v_{k,w} = \begin{cases} 0 & k = 0 \\ v_{k-1,w} & w_k > w \\ \max(v_{k-1,w}, v_{k-1,w-w_k} + b_k) & w_k \leq w \end{cases}$$

- Giải pháp tối ưu của bài toán là giá trị $v_{n,w}$

230

Xếp ba lô

Thuật toán chia để trị (4)

```

balo (k, w)
begin
  if (k = 0) then return 0
  else
    if (wk > w) then
      return balo(k-1, w) // không sử dụng đồ vật k
    else
      x = balo(k-1, w)
      y = balo(k-1, w-wk) + bk
      return (max(x, y))
    endif
  endif
end

```

Sử dụng: balo(n, W)

231

Xếp ba lô

Thuật toán chia để trị (5)

- Đánh giá độ phức tạp
 - Gọi C(n) là độ phức tạp trong trường hợp xấu nhất
 - Từ thuật toán, ta có hệ thức truy hồi

$$C(n) = \begin{cases} c & n = 0 \\ \max(C(n-1), 2C(n-1)) + d & n > 0 \end{cases}$$

$$= \begin{cases} c & n = 0 \\ 2C(n-1) + d & n > 0 \end{cases}$$

với c và d là các hằng số

- Vậy độ phức tạp của thuật toán O(2ⁿ)
 - Hàm mũ !

232

Xếp ba lô

Thuật toán quy hoạch động (1)

- Sử dụng mảng $v[0..n, 0..W]$ để lưu lại các giải pháp của các bài toán con
- $v[k, w]$ là tổng giá trị lớn nhất của ba lô mà trọng lượng không vượt quá w khi chỉ sử dụng các đồ vật 1... k
- Ban đầu
 - $v[0, w] = 0$ với mọi w
 - $v[k, 0] = 0$ với mọi k
- Sau đó, $v[k, w]$ sẽ được tính theo $v[k-1, w]$ hoặc $v[k-1, w-w_k]$

233

Xếp ba lô

Thuật toán quy hoạch động (2)

```

balo (n, W)
begin
  for k from 0 to n do v[k,0] = 0 endfor
  for w from 0 to W do v[0,w] = 0 endfor
  for k from 1 to n do
    for w from 1 to W do
      if ( $w_k \leq w$ ) then // có thể sử dụng đồ vật k
         $v[k, w] = \max(v[k-1, w], v[k-1, w-w_k] + b_k)$ 
      else //  $w_k > w$ 
         $v[k, w] = v[k-1, w]$  // không sử dụng đồ vật k
      endif
    endfor
  endfor
end

```

234

Xếp ba lô

Thuật toán quy hoạch động (3)

```

balo (n, W)
begin
  for k from 0 to n do v[k,0] = 0 endfor
  for w from 0 to W do v[0,w] = 0 endfor
  for k from 1 to n do
    for w from 1 to W do
      if (wk ≤ w) then // có thể sử dụng đồ vật k
        if (bk + v[k-1,w-wk] > v[k-1,w]) then
          v[k,w] = bk + v[k-1,w-wk] // sử dụng đồ vật k
        else
          v[k,w] = v[k-1,w] // không sử dụng đồ vật k
        endif
      else // wk > w
        v[k,w] = v[k-1,w] // không sử dụng đồ vật k
      endif
    endfor
  endfor
end

```

235

Xếp ba lô

Thuật toán quy hoạch động (4)

- Đánh giá độ phức tạp
 - Thuật toán tính mảng $n \times W$ phần tử bởi hai vòng lặp lồng nhau
 - Độ phức tạp $O(nW)$
- Ví dụ minh họa
 - $W = 5$ (trọng lượng tối đa ba lô có thể chứa)
 - $n = 4$ (4 đồ vật)
 - Các đồ vật lần lượt có (trọng lượng, giá trị):
(2,3), (3,4), (4,5), (5,6)

236

Xếp ba lô

Ví dụ (2)

	k	0	1	2	3	4
W						
0		0				
1		0				
2		0				
3		0				
4		0				
5		0				

for w = 0 to W do v[0,w] = 0

237

Xếp ba lô

Ví dụ (3)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0				
2		0				
3		0				
4		0				
5		0				

for k = 0 to n do v[k,0] = 0

238

Xếp ba lô

Ví dụ (4)

W	k	0	1	2	3	4	Đồ vật
0		0	0	0	0	0	1: (2,3)
1		0 → 0					2: (3,4)
2		0					3: (4,5)
3		0					4: (5,6)
4		0					
5		0					

$k=1$
 $b_k=3$
 $w_k=2$
 $w=1$
 $w-w_k=-1$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

239

Xếp ba lô

Ví dụ (5)

W	k	0	1	2	3	4	Đồ vật
0		0	0	0	0	0	1: (2,3)
1		0	0				2: (3,4)
2		0	3				3: (4,5)
3		0					4: (5,6)
4		0					
5		0					

$k=1$
 $b_k=3$
 $w_k=2$
 $w=2$
 $w-w_k=0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

240

Xếp ba lô

Ví dụ (6)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0				2: (3,4)
2	0	3				3: (4,5)
3	0	3				4: (5,6)
4	0					
5	0					

$k=1$
 $b_k=3$
 $w_k=2$
 $w=3$
 $w-w_k=1$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

241

Xếp ba lô

Ví dụ (7)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0				2: (3,4)
2	0	3				3: (4,5)
3	0	3				4: (5,6)
4	0	3				
5	0					

$k=1$
 $b_k=3$
 $w_k=2$
 $w=4$
 $w-w_k=2$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

242

Xếp ba lô

Ví dụ (8)

W	k	0	1	2	3	4	Đồ vật
0		0	0	0	0	0	1: (2,3)
1		0	0				2: (3,4)
2		0	3				3: (4,5)
3		0	3				4: (5,6)
4		0	3				
5		0	3				

$k=1$

$b_k=3$

$w_k=2$

$w=5$

$w-w_k=2$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

243

Xếp ba lô

Ví dụ (9)

W	k	0	1	2	3	4	Đồ vật
0		0	0	0	0	0	1: (2,3)
1		0	0	0			2: (3,4)
2		0	3				3: (4,5)
3		0	3				4: (5,6)
4		0	3				
5		0	3				

$k=2$

$b_k=4$

$w_k=3$

$w=1$

$w-w_k=-2$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

244

Xếp ba lô

Ví dụ (10)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0			2: (3,4)
2	0	3	→ 3			3: (4,5)
3	0	3				4: (5,6)
4	0	3				
5	0	3				

$k=2$

$b_k=4$

$w_k=3$

$w=2$

$w-w_k=-1$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

245

Xếp ba lô

Ví dụ (11)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0			2: (3,4)
2	0	3	3			3: (4,5)
3	0	3	4			4: (5,6)
4	0	3				
5	0	3				

$k=2$

$b_k=4$

$w_k=3$

$w=3$

$w-w_k=0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

246

Xếp ba lô

Ví dụ (12)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0			2: (3,4)
2	0	3	3			3: (4,5)
3	0	3	4			4: (5,6)
4	0	3	4			
5	0	3				

$k=2$
 $b_k=4$
 $w_k=3$
 $w=4$
 $w-w_k=1$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

247

Xếp ba lô

Ví dụ (13)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0			2: (3,4)
2	0	3	3			3: (4,5)
3	0	3	4			4: (5,6)
4	0	3	4			
5	0	3	7			

$k=2$
 $b_k=4$
 $w_k=3$
 $w=5$
 $w-w_k=2$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

248

Xếp ba lô

Ví dụ (14)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0	0		2: (3,4)
2	0	3	3	3		3: (4,5)
3	0	3	4	4		4: (5,6)
4	0	3	4			
5	0	3	7			

$k=3$

$b_k=5$

$w_k=4$

$w=1..3$

$w - w_k < 0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

249

Xếp ba lô

Ví dụ (15)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0	0		2: (3,4)
2	0	3	3	3		3: (4,5)
3	0	3	4	4		4: (5,6)
4	0	3	4	5		
5	0	3	7			

$k=3$

$b_k=5$

$w_k=4$

$w=4$

$w - w_k = 0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

250

Xếp ba lô

Ví dụ (16)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0	0		2: (3,4)
2	0	3	3	3		3: (4,5)
3	0	3	4	4		4: (5,6)
4	0	3	4	5		
5	0	3	7	→ 7		

$k=3$

$b_k=5$

$w_k=4$

$w=5$

$w-w_k=1$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

251

Xếp ba lô

Ví dụ (17)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0	0	→ 0	2: (3,4)
2	0	3	3	3	→ 3	3: (4,5)
3	0	3	4	4	→ 4	4: (5,6)
4	0	3	4	5	→ 5	
5	0	3	7	7		

$k=4$

$b_k=6$

$w_k=5$

$w=1..4$

$w-w_k < 0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

252

Xếp ba lô

Ví dụ (18)

W \ k	0	1	2	3	4	Đồ vật
0	0	0	0	0	0	1: (2,3)
1	0	0	0	0	0	2: (3,4)
2	0	3	3	3	3	3: (4,5)
3	0	3	4	4	4	4: (5,6)
4	0	3	4	5	5	
5	0	3	7	7	7	

$k=4$

$b_k=6$

$w_k=5$

$w=5$

$w-w_k=0$

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

253

Xếp ba lô

- Thuật toán *ballo* chỉ mới tìm được tổng giá trị lớn nhất mà ba lô có thể chứa

Bài tập

- Xây dựng giải pháp tối ưu
 - Các vật được chứa trong ba lô
- Minh hoạ

Các đồ vật
sử dụng: 1, 2

W \ k	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	7

254

Bài tập

□ Bài 1

- Số Fibonacci được định nghĩa

$$F_0 = 1, F_1 = 1$$

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

Xây dựng thuật toán quy hoạch động tính F_n

□ Bài 2

- Số Catalan được định nghĩa

$$T(n) = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-1} T(i).T(n-i) & n > 1 \end{cases}$$

Xây dựng thuật toán quy hoạch động tính T_n

255

Bài tập

□ Bài 3

Một băng cát-xét gồm 2 mặt, mỗi mặt có thể ghi được d phút (chẳng hạn $d=30$). Một đĩa CD chứa n bài hát có tổng thời lượng m phút ($m > d$, chẳng hạn $m=78$). Bài hát i có thời lượng d_i phút, $d_i > 0$. Cần chọn các bài hát từ đĩa CD ghi lên các mặt băng cát-xét sao cho tổng thời lượng là lớn nhất. Một bài hát được ghi lên một mặt đĩa hoặc không được ghi.

Gọi $time(i, t_1, t_2)$ là thời lượng lớn nhất có thể ghi lên băng cát-xét, trong đó t_1 (t_2) là thời lượng lớn nhất có thể ghi lên mặt thứ nhất (mặt thứ hai), và chỉ sử dụng i bài hát đầu tiên. Khi đó $time(n, d, d)$ là giải pháp của bài toán.

Hãy thực hiện:

1. Xây dựng hệ thức truy hồi tính $time(i, t_1, t_2)$.
2. Xây dựng thuật toán đệ quy tính hệ thức truy hồi trên.
3. Đánh giá độ phức tạp thuật toán đệ quy tính hệ thức truy hồi trên (giả thiết $d_i = 1$ với mọi i).
4. Xây dựng thuật toán quy hoạch động.
5. Đánh giá độ phức tạp thuật toán quy hoạch động.

256

Giải thích 1

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n \geq 2 \end{cases}$$

□ Chứng minh $P(n) = \Omega(2^n)$

■ Chứng minh bằng quy nạp $P(n) \geq 2^{n-2}$

□ $P(n)$ đúng với $n \leq 4$

□ Với $n \geq 5$

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &\geq \sum_{k=1}^{n-1} 2^{k-2} 2^{n-k-2} = \sum_{k=1}^{n-1} 2^{n-4} = (n-1)2^{n-4} \\ &\geq 2^{n-2} \quad (\text{do } n \geq 5) \end{aligned}$$

■ Vậy $P(n) = \Omega(2^n)$



257

Thuật toán tham lam (7)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Thuật toán tham lam (greedy algorithms)

- Vấn đề tìm kiếm giải pháp tối ưu
 - Chia bài toán thành nhiều bài toán con
 - Giải quyết các bài toán con
 - Giải pháp của các bài toán con sẽ là giải pháp cho bài toán đặt ra
- Thuật toán chia để trị hoặc thuật toán đơn giản
 - Giải quyết **tất cả các bài toán con**
 - Độ phức tạp cao (thường hàm mũ)
 - Dễ thiết kế, dễ cài đặt
- Thuật toán quy hoạch động
 - **Ghi nhớ lại giải pháp của các bài toán con** (khi các bài toán con không hoàn toàn độc lập) để tránh các xử lý trùng lặp
 - Độ phức tạp thấp hơn (thường hàm đa thức)
 - Tuy nhiên, khó thiết kế và cài đặt giải pháp

259

Thuật toán tham lam

- Nguyên tắc thuật toán tham lam
 - Tối ưu từng bước -> tối ưu toàn cục
 - Chỉ giải quyết một bài toán con
 - Không xét tất cả các bài toán con
 - Xây dựng giải pháp từng bước một
 - Ở mỗi bước, thực hiện sự chọn lựa tốt nhất tại thời điểm đó (giải pháp cục bộ)
 - Không có giải pháp tổng thể
 - Không quay lại xem xét quyết định đã chọn lựa
 - Hy vọng kết quả thu được là giải pháp tối ưu

260

Thuật toán tham lam

□ Ưu điểm

- dễ thiết kế
- dễ cài đặt
- độ phức tạp thấp

□ Nhược điểm

- Không phải luôn cho giải pháp tối ưu
- Khó để chứng minh thuật toán cho giải pháp tối ưu

261

Thuật toán tham lam

□ Cấu trúc tổng quát

```

thamlam(C: tập hợp các ứng cử viên)
// hàm trả về giải pháp tối ưu, gồm các ứng cử viên
begin
    S = ∅           // S là giải pháp tối ưu
    while (C ≠ ∅ và S chưa là giải pháp) do
        x = chọn(C) // chọn x từ tập C theo tiêu chí của hàm chọn
        C = C - {x}
        if (S ∪ {x} có triển vọng là giải pháp) then S := S ∪ {x}
    endif
endwhile
if (S là lời giải) then return S
else return 0
endif
end

```

262

Một số ứng dụng

- ▣ Thuê xe
- ▣ Đổi tiền
- ▣ Xếp ba lô
- ▣ Mã Huffman
- ▣ Tìm cây khung nhỏ nhất
 - Thuật toán Kruskal
 - Thuật toán Prim
- ▣ Tìm đường đi ngắn nhất
 - Thuật toán Dijkstra

263

Thuê xe

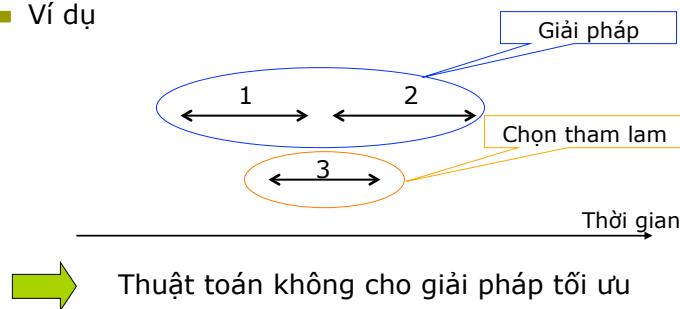
- ▣ Bài toán
 - Có một chiếc xe ô tô duy nhất và có nhiều khách hàng yêu cầu được thuê xe. Mỗi yêu cầu thuê xe có một thời điểm bắt đầu thuê và một thời điểm kết thúc thuê.
 - Vấn đề: sắp xếp việc cho thuê làm sao để số khách hàng được thuê xe là nhiều nhất
- ▣ Hình thức hoá
 - Giả sử $S = \{a_1, a_2, \dots, a_n\}$ tập hợp các yêu cầu thuê xe
 - Mỗi $a_i \in S$, $s(a_i)$ là thời điểm bắt đầu thuê và $f(a_i)$ là thời điểm kết thúc thuê
 - Gọi F là tập hợp lớn nhất các yêu cầu thoả mãn:
 - ▣ Hai yêu cầu bất kỳ phải lệch nhau về thời gian, nghĩa là yêu cầu tiếp theo chỉ được bắt đầu khi yêu cầu trước đó kết thúc
 - ▣ Hay: $\forall a_1 \in F, a_2 \in F: s(a_1) \leq s(a_2) \Rightarrow f(a_1) \leq s(a_2)$

264

Thuê xe

□ Phép thử 1

- Sắp xếp các yêu cầu thuê xe theo thứ tự tăng dần thời gian thuê
- Chọn tham lam
 - chọn yêu cầu có thời gian thuê ngắn nhất
- Ví dụ

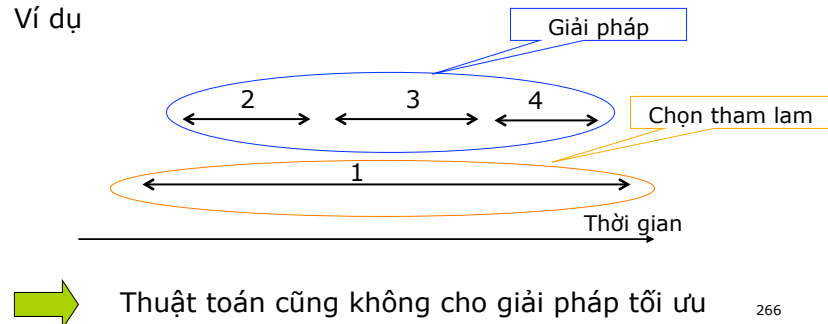


265

Thuê xe

□ Phép thử 2

- Sắp xếp các yêu cầu thuê xe theo thứ tự thời điểm bắt đầu thuê
- Chọn tham lam
 - chọn yêu cầu có thời điểm bắt đầu thuê sớm nhất
- Ví dụ



266

Thuê xe

□ Phép thử 3

- Sắp xếp các yêu cầu theo thứ tự thời điểm kết thúc thuê tăng dần
- Chọn tham lam
 - chọn yêu cầu có thời điểm kết thúc thuê sớm nhất
- Thuật toán cho giải pháp tối ưu
 - Tại sao ?
 - Chứng minh !

267

Thuê xe

□ Chứng minh tính tối ưu của thuật toán

- Giả sử $F = \{x_1, x_2, \dots, x_p\}$ là giải pháp đạt được bởi thuật toán tham lam và $G = \{y_1, y_2, \dots, y_q\}$ với $q \geq p$ là một giải pháp tối ưu (cho phép thực hiện nhiều yêu cầu nhất)
- Cần chứng minh F là giải pháp tối ưu, nghĩa là $p = q$
- Giả sử các phần tử của các tập hợp F và G được sắp xếp theo thứ tự thời điểm kết thúc thuê tăng dần
- Nếu G không chứa F , thì phải tồn tại k sao cho: $\forall i < k, x_i = y_i$ và $x_k \neq y_k$. x_k được chọn bởi thuật toán tham lam, nên x_k có $f(x_k)$ nhỏ nhất và thời điểm bắt đầu sau $x_{k-1} = y_{k-1}$. Vậy $f(x_k) \leq f(y_k)$. Thay G bởi $G' = \{y_1, y_2, \dots, y_{k-1}, x_k, y_{k+1}, \dots, y_q\}$ thỏa mãn ràng buộc sự lênh nhau về thời gian của các yêu cầu. Vậy G' cũng là một giải pháp tối ưu mà có số yêu cầu trùng với F nhiều hơn so với G .
- Lặp lại bước trên, cuối cùng có được G'' chứa F mà $|G''| = |G|$
- Nếu G'' có chứa yêu cầu không thuộc F (tức là các yêu cầu bắt đầu sau khi x_p kết thúc) thì yêu cầu đó đã phải được thêm vào F theo thuật toán tham lam
- Vậy $G'' = F$, mà $|G''| = |G|$, nên F là giải pháp tối ưu

268

Thuê xe

▣ Thuật toán

```

thuexe(S)
begin
  n = length(S)
  // Sắp xếp các yêu cầu theo thời điểm kết thúc thuê tăng dần
  S = {a1, a2, ..., an} với f(a1) ≤ f(a2) ≤ ... ≤ f(an)
  F = {a1}
  i = 1
  for j from 2 to n do
    if (f(ai) ≤ s(aj)) then
      F = F ∪ {aj}
      i = j
    endif
  endfor
  return F
end

```

Thuê xe

▣ Độ phức tạp của thuật toán

- Sắp xếp các yêu cầu
 - ▣ $O(n \log n)$
- Xây dựng giải pháp
 - ▣ $O(n)$

Tính chất của chiến lược tham lam

- Thuật toán tham lam
 - Xác định giải pháp sau một dãy liên tiếp các lựa chọn
 - Mỗi bước quyết định, lựa chọn đường như tốt nhất ở bước đó sẽ được chọn
 - Không luôn cho giải pháp tối ưu
- Một bài toán tối ưu bất kỳ có thể được giải quyết bởi thuật toán tham lam ?
 - Không luôn luôn đúng
- Tuy nhiên, nếu một bài toán có hai tính chất
 - Tính chất chọn lựa tham lam
 - Cấu trúc con tối ưu
- Thì bài toán có thể giải quyết được bởi thuật toán tham lam

271

Tính chất của chiến lược tham lam

- Tính chất chọn lựa tham lam (greedy choice property)
 - Luôn tồn tại một giải pháp tối ưu chứa một chọn lựa tham lam
 - Cần chỉ ra tồn tại một giải pháp tối ưu luôn bắt đầu bởi một chọn lựa tham lam
- Tính chất cấu trúc con tối ưu (optimal substructure)
 - Nếu F là một giải pháp tối ưu chứa một chọn lựa tham lam c thì $F - \{c\}$ là giải pháp tối ưu cho bài toán con tương tự như bài toán đầu không chứa c
 - Giải pháp tối ưu của một bài toán *phải* chứa giải pháp tối ưu của bài toán con của nó



Chứng minh tính tối ưu của thuật toán tham lam

272

Tính chất của chiến lược tham lam

- Nếu một bài toán thoả mãn hai tính chất
 - Tính chất chọn lựa tham lam
 - Tính chất cấu trúc con tối ưu
- Thì thuật toán tham lam cho giải pháp tối ưu
- Chứng minh
 - Theo tính chất chọn lựa tham lam, tồn tại giải pháp tối ưu F chứa một chọn lựa tham lam c_1 . Theo tính chất cấu trúc con tối ưu, $F - \{c_1\}$ là giải pháp tối ưu của bài toán con không chứa c_1 .
 - Áp dụng cho bài toán con không chứa c_1 , theo tính chất chọn lựa tham lam, $F - \{c_1\}$ là giải pháp tối ưu chứa chọn lựa tham lam c_2 . Theo tính chất cấu trúc con tối ưu, $F - \{c_1, c_2\}$ là giải pháp tối ưu cho bài toán con không chứa c_1 và c_2 .
 - Tiếp tục lý giải như thế, cuối cùng chúng ta có

$$F - \{c_1, c_2, \dots, c_n\} = \emptyset$$
 - Hay: $F = \{c_1, c_2, \dots, c_n\}$
 - Vậy giải pháp tối ưu F của bài toán ban đầu là một dãy các lựa chọn tham lam thực hiện bởi thuật toán tham lam

273

Chứng minh tính tối ưu thuật toán tham lam

- 2 cách
 - Chứng minh trực tiếp giải pháp của thuật toán là tối ưu
 - Nghĩa là không tồn tại giải pháp tối ưu khác tốt hơn
 - Chứng minh bài toán thoả mãn hai tính chất
 - Tính chất chọn lựa tham lam
 - Tính chất cấu trúc con tối ưu

274

Đổi tiền

□ Bài toán

- Cho một hệ thống tiền tệ gồm các loại tờ giấy tiền có mệnh giá là 1, 5, 10, 20, 50. Cần đổi một số tiền S sao cho số tờ cần dùng ít nhất.
- Ví dụ
 - $98 = 1 + 1 + 1 + 5 + 50 + 20 + 20$
- Thuật toán đơn giản
 - liệt kê tất cả các kết hợp có thể cho tổng số tiền là S
 - chọn kết hợp dùng ít số tờ nhất
 - Độ phức tạp hàm mũ !

275

Đổi tiền

□ Thuật toán tham lam

- Chọn lựa tham lam
 - ở mỗi bước, chọn tờ giấy tiền có mệnh giá cao nhất có thể mà không vượt quá tổng số tiền cần đổi
- Ví dụ: $S = 98$

$S=98$	$S=48$	$S=28$	$S=8$	$S=3$	$S=2$	$S=1$
50	20	20	5	1	1	1

276

Đổi tiền

- Chứng minh thuật toán tham lam cho giải pháp tối ưu (1)
 - Giả sử F là giải pháp tối ưu
 - F phải thoả mãn các ràng buộc
 - F chỉ chứa nhiều nhất 4 tờ tiền mệnh giá 1
 - 5 tờ mệnh giá 1 = 1 tờ mệnh giá 5
 - F chỉ chứa nhiều nhất 1 tờ tiền mệnh giá 5
 - 2 tờ mệnh giá 5 = 1 tờ mệnh giá 10
 - F chỉ chứa nhiều nhất 1 tờ tiền mệnh giá 10
 - 2 tờ mệnh giá 10 = 1 tờ mệnh giá 20
 - Nếu F không chứa tờ tiền nào mệnh giá 10, thì chỉ chứa nhiều nhất 2 tờ mệnh giá 20
 - 3 tờ mệnh giá 20 = 1 tờ mệnh giá 50 + 1 tờ mệnh giá 10
 - Nếu F có chứa tờ tiền mệnh giá 10, thì chỉ chứa nhiều nhất 1 tờ mệnh giá 20
 - 2 tờ mệnh giá 20 + 1 tờ mệnh giá 10 = 1 tờ mệnh giá 50

277

Đổi tiền

- Chứng minh thuật toán tham lam cho giải pháp tối ưu (2)
 - Chỉ cần chỉ ra bài toán thoả mãn hai tính chất
 - **Tính chất chọn lựa tham lam:** cần chỉ ra luôn tồn tại giải pháp tối ưu bắt đầu bởi một chọn lựa tham lam
 - Nếu $S \geq 50$, thuật toán tham lam chọn lựa tờ tiền mệnh giá 50. Cần chứng minh rằng F phải bắt đầu bởi chọn lựa tờ tiền mệnh giá 50. Bằng phản chứng, giả sử F không chọn lựa tờ tiền mệnh giá 50. Vì F phải thoả mãn các ràng buộc trên, nên có thể chỉ có 2 khả năng: $4 \times 1 + 5 + 10 + 20 < 50$ và $4 \times 1 + 5 + 2 \times 20 < 50$.
 Vậy nếu $S \geq 50$, F phải chứa ít nhất một tờ tiền mệnh giá 50
 - Nếu $50 > S \geq 20$, lý giải tương tự, F phải chứa ít nhất một tờ tiền mệnh giá 20
 - Tiếp tục ...

278

Đổi tiền

□ Chứng minh thuật toán tham lam cho giải pháp tối ưu (3)

■ Tính chất cấu trúc con tối ưu

- Giả sử F là giải pháp tối ưu cho tổng số tiền S , p là tờ tiền được chọn lựa cuối cùng bởi thuật toán tham lam. Cần chỉ ra rằng $F - \{p\}$ là giải pháp tối ưu cho bài toán con $S - p$.
- Chứng minh bằng phản chứng: giả sử tồn tại một giải pháp tối ưu tốt hơn F' cho bài toán con $S - p$. Khi đó, $F' \cup \{p\}$ là giải pháp tối ưu tốt hơn F cho bài toán S . Điều này mâu thuẫn giả thiết.

Vậy $F' \cup \{p\} = F$ hay $F' = F - \{p\}$.

279

Đổi tiền

□ Thuật toán tham lam

```
doitien(S)
begin
  F = ∅
  if (S ≥ 50) then
    F = F ∪ {(S div 50) tờ mệnh giá 50}
    S = S mod 50
  endif
  if (S ≥ 20) then
    F = F ∪ {(S div 20) tờ mệnh giá 20}
    S = S mod 20
  endif
  if (S ≥ 10) then
    F = F ∪ {(S div 10) tờ mệnh giá 10}
    S = S mod 10
  endif
  // tương tự cho các tờ tiền mệnh giá 5, 2, 1
  ...
end
```

280

Đổi tiền

□ Lưu ý

- Thuật toán tham lam này không cho giải pháp tối ưu đối với mọi hệ thống tiền tệ
 - Chẳng hạn, thuật toán sẽ không cho giải pháp tối ưu đối với hệ thống tiền tệ $\{6, 4, 1\}$
 - Ví dụ
 - $S = 8$
 - Giải pháp cho bởi thuật toán tham lam: $6 + 1 + 1$
 - Giải pháp tối ưu: $4 + 4$

281

Xếp ba lô

□ Bài toán

- cho n đồ vật và một ba lô có trọng lượng tối đa W
- mỗi đồ vật i có trọng lượng w_i
- mỗi đồ vật i có giá trị v_i
- gọi x_i là một phần của đồ vật i , $0 \leq x_i \leq 1$, x_i có trọng lượng $x_i w_i$ và giá trị $x_i v_i$
- Yêu cầu: xếp các đồ vật vào ba lô để tổng giá trị ba lô lớn nhất

- Bài toán xếp ba lô này được gọi là *xếp ba lô « từng phần »*
 - có thể chỉ cần xếp vào ba lô một phần của đồ vật

- Bài toán xếp ba lô đã gặp được gọi là *xếp ba lô « 0-1 »*
 - một đồ vật hoặc được xếp vào ba lô (1) hoặc không được xếp vào ba lô (0)

282

Xếp ba lô

□ Ý tưởng

- Tập ứng cử viên là các đồ vật
- Ở mỗi bước, chọn đồ vật *triển vọng nhất* và xếp vào ba lô một phần lớn nhất có thể của đồ vật này
 - đối với các đồ vật được chọn đầu tiên, xếp toàn bộ đồ vật vào ba lô
 - đối với đồ vật được chọn cuối cùng, có thể chỉ xếp một phần đồ vật vào ba lô
- Thuật toán dừng khi ba lô đầy
- Chọn đồ vật theo tiêu chí nào ?
 - Giá trị giảm dần
 - Trọng lượng tăng dần
 - Tỷ lệ giá trị trên trọng lượng (v_i/w_i) giảm dần
- **Chọn lựa tham lam:** chọn đồ vật có tỷ lệ giá trị trên trọng lượng (v_i/w_i) giảm dần

283

Xếp ba lô

□ Thuật toán

```

xepbalotungphan()
begin
  sắp xếp các đồ vật theo tỷ lệ  $v_i/w_i$  giảm dần
   $w = W$ 
   $i = 1$ 
  while ( $w \geq w_i$ ) do // xếp toàn bộ đồ vật vào ba lô
     $x_i = 1$ 
     $w = w - w_i$ 
     $i = i + 1$ 
  endwhile
   $x_i = w/w_i$  // đồ vật cuối cùng được chọn để xếp vào ba lô
  for k from  $i + 1$  to  $n$  do
     $x_i = 0$  // các đồ vật không được xếp vào ba lô
  endfor
  return ( $x_1, x_2, \dots, x_n$ ) // giải pháp
end

```

284

Xếp ba lô

Thuật toán

- Phân tích độ phức tạp
 - Sắp xếp: $O(n \log n)$
 - Lặp: duyệt n đồ vật, vậy $O(n)$

285

Xếp ba lô

Chứng minh tính tối ưu (1)

■ Cách 1: chứng minh trực tiếp (1)

- Giả sử $v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n$
- $X = \{x_1, x_2, \dots, x_n\}$ là giải pháp được xác định bởi thuật toán tham lam, $V(X)$ là tổng giá trị
- Giả sử j là chỉ số nhỏ nhất mà $x_j < 1$, vậy $X = \{1, \dots, 1, x_j, 0, \dots, 0\}$ (chỉ có đồ vật cuối cùng được chọn một số phần)
- Bằng phản chứng, giả sử $Y = \{y_1, y_2, \dots, y_n\}$ giải pháp khác và $V(Y)$ là tổng giá trị
- Ta có

$$\begin{aligned} V(X) - V(Y) &= \sum_{i=1}^n x_i v_i - \sum_{i=1}^n y_i v_i = \sum_{i=1}^n v_i (x_i - y_i) \\ &= \sum_{i=1}^n (x_i - y_i) w_i \frac{v_i}{w_i} \end{aligned}$$

286

Xếp ba lô

□ Chứng minh tính tối ưu (2)

■ Cách 1: chứng minh trực tiếp (2)

□ Có ba trường hợp xảy ra

- $i < j$, thì $x_i = 1$, vậy $x_i - y_i \geq 0$ và $v_i/w_i \geq v_j/w_j$, suy ra

$$(x_i - y_i)v_i/w_i \geq (x_i - y_i)v_j/w_j$$

- $i > j$, thì $x_i = 0$, vậy $x_i - y_i \leq 0$ và $v_i/w_i \leq v_j/w_j$, cũng suy ra

$$(x_i - y_i)v_i/w_i \geq (x_i - y_i)v_j/w_j$$

- $i = j$, thì $(x_i - y_i)v_i/w_i = (x_i - y_i)v_j/w_j$

Vậy ta có:

$$V(X) - V(Y) \geq \frac{v_j}{w_j} \sum_{i=1}^n (x_i - y_i)w_i = \frac{v_j}{w_j} \left(W - \sum_{i=1}^n y_i w_i \right) \geq 0 \text{ do } \sum_{i=1}^n y_i w_i \leq W$$

Như thế: $V(X) \geq V(Y)$

Hay $V(X)$ là giải pháp tối ưu

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

287

Xếp ba lô

□ Chứng minh tính tối ưu (3)

■ Cách 2: chứng minh hai tính chất

□ **Tính chất chọn lựa tham lam:** cần chỉ ra rằng tồn tại giải pháp tối ưu chứa một chọn lựa tham lam

■ Giả sử k là đồ vật có tỉ lệ v_k/w_k lớn nhất, S là một giải pháp tối ưu, giá trị của S là $V(S)$

■ Bằng phản chứng, giả sử S không chứa k

■ Tổng giá trị của S là $V(S) = \sum_{i=1}^n x_i v_i$

■ Nếu một số phần của đồ vật k còn lại không được chọn, thì khi đó, với $j \in S$, $x_j \neq 0$, $j \neq k$, thay thế j trong S bởi k , với cùng trọng lượng $x_j w_j = x_k w_k$ (để không vượt trọng lượng ba lô), ta sẽ nhận được giải pháp S' tốt hơn S . Mâu thuẫn giả thiết.

$$x_j v_j \leq x_k v_k \quad \text{chia cho } x_j w_j = x_k w_k \Rightarrow \frac{v_j}{w_j} \leq \frac{v_k}{w_k}$$

➡ S phải chứa đồ vật k

288

Xếp ba lô

□ Chứng minh tính tối ưu (4)

■ Cách 2: chứng minh hai tính chất

- **Tính chất cấu trúc con tối ưu:** nếu S là giải pháp tối ưu chứa chọn lựa tham lam c thì tồn tại giải pháp S' tối ưu cho bài toán con không chứa c
 - Giả sử S là giải pháp tối ưu chứa đồ vật k có tỷ lệ v_k/w_k lớn nhất với trọng lượng lớn nhất có thể ($p = \max(W, w_k \text{ còn lại})$)
 - Khi đó $S' = S - \{k\}$ là giải pháp cho bài toán con không chứa k với ba lô trọng lượng tối đa giảm p
 - Bằng phản chứng, giả sử S' không tối ưu. Khi đó tồn tại S'' tốt hơn S' là giải pháp tối ưu cho bài toán con không chứa k . Vậy, $S' \cup \{k\}$, với k có trọng lượng p sẽ là giải pháp tốt hơn giải pháp S cho bài toán ban đầu. Mâu thuẫn giả thiết.
 - S' phải là giải pháp tối ưu cho bài toán con.

289

Xếp ba lô

□ Lưu ý

- Bài toán xếp ba lô từng phần được giải quyết bởi thuật toán tham lam
- Bài toán xếp ba lô 0-1 không thể được giải quyết bởi thuật toán tham lam
 - Ngược lại, được giải quyết bởi thuật toán quy hoạch động

290

Mã Huffman

- ▣ Kỹ thuật hiệu quả trong nén dữ liệu
 - tiết kiệm từ 20 đến 90% không gian lưu trữ
- ▣ Ý tưởng
 - Xem dữ liệu là một dãy kí tự
 - Sử dụng tần suất xuất hiện của mỗi kí tự để xây dựng giải pháp tối ưu bằng cách biểu diễn mỗi kí tự bởi một xâu nhị phân
 - Thuật toán tham lam

291

Mã Huffman

- ▣ Ví dụ
 - Cần nén một tệp chứa 100.000 kí tự, tệp chỉ gồm các kí tự a, b, c, d, e, f. Tần suất xuất hiện của các kí tự trong tệp được xác định
 - Biểu diễn mỗi kí tự bằng xâu nhị phân có độ dài hằng số: *mã có độ dài hằng số* (fixed-length codes)
 - ▣ Cần 3 bit để biểu diễn 6 kí tự
- | Kí tự | a | b | c | d | e | f |
|----------|-------|-------|-------|-------|------|------|
| Tần suất | 45000 | 13000 | 12000 | 16000 | 9000 | 5000 |
| Dãy bit | 000 | 001 | 010 | 011 | 100 | 101 |
- Số bit cần thiết để lưu trữ tệp: 300000
 - Tồn tại giải pháp tốt hơn ?

292

Mã Huffman

- Mỗi kí tự được biểu diễn bằng độ dài xâu nhị phân thay đổi được: *mã có độ dài thay đổi* (variable-length codes)
 - Kí tự có tần suất xuất hiện lớn được biểu diễn bởi xâu nhị phân có độ dài ngắn hơn
- Ví dụ

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bit	0	101	100	111	1101	1100
- Số bit cần để lưu trữ tệp
 $(45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 4 + 9 \times 4 + 5 \times 4) \times 1000 = 224000$ bit
- Chính là giải pháp tối ưu cho tệp dữ liệu này

293

Mã Huffman

- Mã tiền tố (prefix codes)
 - Không một mã nào là tiền tố của một mã khác
 - Quá trình giải mã trở nên đơn giản
 - Xác định mã của mỗi kí tự không nhập nhằng
- Ví dụ
 - Với bảng mã

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bit	0	101	100	111	1101	1100
 - Dãy kí tự *abc* được mã hoá: *0101100*
 - Chuỗi nhị phân *001011101* được giải mã một cách duy nhất: *aabe*

294

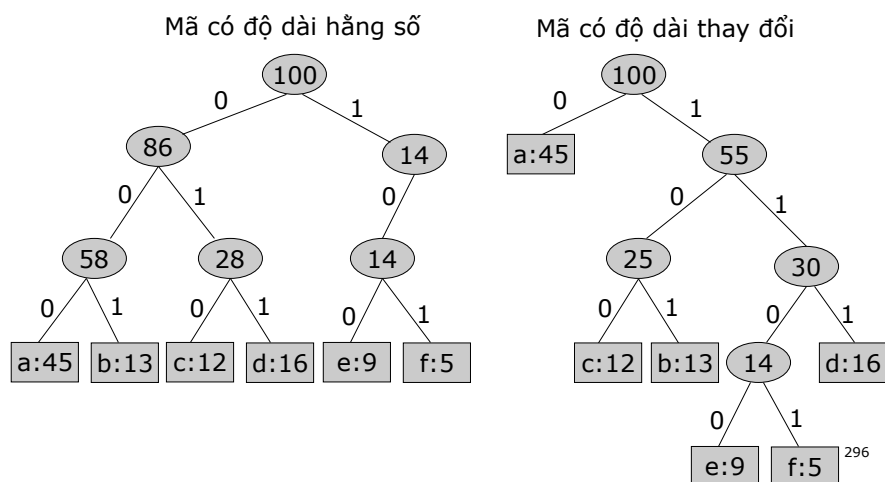
Mã Huffman

- ▣ Quá trình giải mã cần một **sự biểu diễn các mã tiền tố** để xác định mỗi mã có thể được xác định dễ dàng
 - **Cây nhị phân** với mỗi nút lá biểu diễn một ký tự
 - Mã nhị phân cho mỗi ký tự là đường đi từ nút gốc đến nút lá chứa ký tự đó
 - ▣ đọc 1 nghĩa là đi đến nút phải, đọc 0 đi đến nút trái
 - Cây nhị phân gồm
 - ▣ Mỗi nút lá được gán nhãn là một ký tự và tần suất xuất hiện của ký tự đó
 - ▣ Mỗi nút trong được gán nhãn là tổng tần suất xuất hiện của các nút lá của các cây con của nút trong đó

295

Mã Huffman

- ▣ Ví dụ cây nhị phân




Mã Huffman

□ Nhận xét

- Mã có độ dài thay đổi, cũng là mã tối ưu, được biểu diễn bởi *cây nhị phân đầy đủ*
 - Cây nhị phân đầy đủ là cây mà mỗi nút trong có đúng hai nút con
- Mã có độ dài cố định, là mã không tối ưu, được biểu diễn bởi cây nhị phân không đầy đủ

□ Chỉ xét mã độ dài thay đổi, nghĩa là chỉ làm việc với cây nhị phân đầy đủ

- Gọi C là bảng kí tự được sử dụng bởi các tệp dữ liệu
- Số nút lá của cây nhị phân là |C|
 - Mỗi nút lá biểu diễn một kí tự
- Số nút trong của cây nhị phân |C| - 1
 - Chứng minh 

297

Mã Huffman

□ Tính số bit cần để mã hoá tệp

- Giả sử T là cây nhị phân biểu diễn các mã tiền tố
- Mỗi kí tự $c \in C$
 - $f(c)$ là tần suất xuất hiện của kí tự c
 - $d(c)$ là độ sâu của nút lá biểu diễn c, chính là số bit biểu diễn c
- Số bit cần để mã hoá tệp là

$$B(T) = \sum_{c \in C} f(c)d(c)$$

- $B(T)$ được gọi là chi phí của cây T

298

Mã Huffman

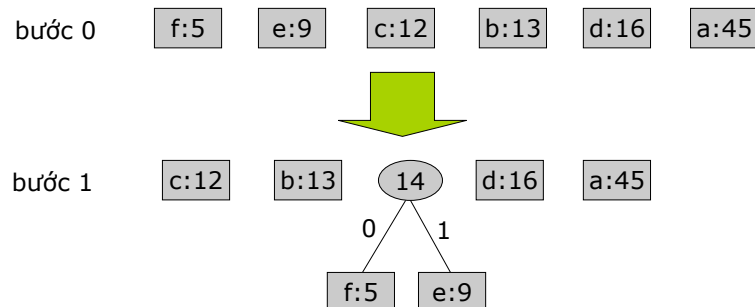
Thuật toán tham lam phát minh bởi Huffman

- Xây dựng cây nhị phân mã tiền tố tối ưu, hay còn được gọi là mã Huffman
 - Bắt đầu bởi $|C|$ nút lá và thực hiện $|C|-1$ phép hoà nhập các cây con để tạo ra cây nhị phân cuối cùng
 - Tại sao $|C| - 1$ phép hoà nhập ?
 - Chọn lựa tham lam: ở mỗi bước, hai cây con có tần suất xuất hiện thấp nhất được chọn để hoà nhập
 - Khi hoà nhập hai cây con, một cây con mới được tạo ra với tần suất xuất hiện bằng tổng tần suất xuất hiện của hai cây con được hoà nhập

299

Mã Huffman

Ví dụ (1)

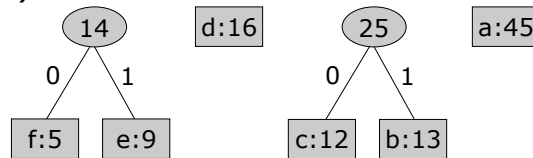


300

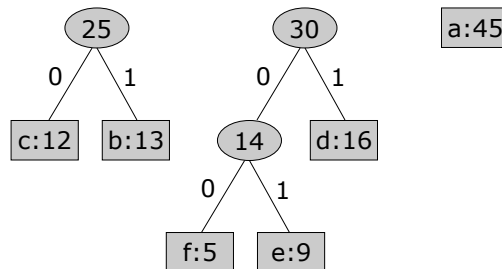
Mã Huffman

▣ Ví dụ (2)

bước 3



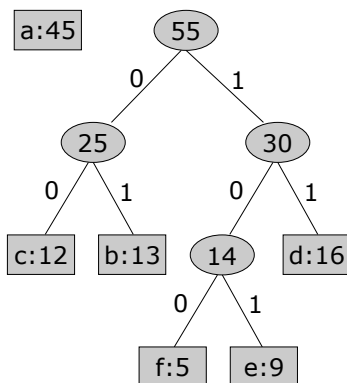
bước 4



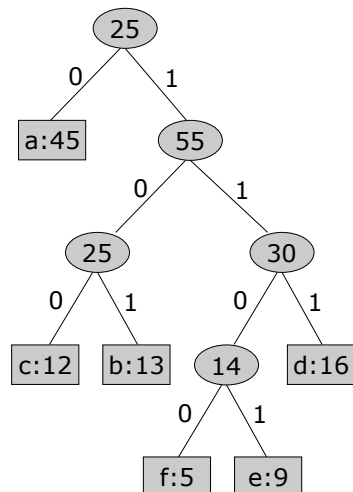
301

Mã Huffman

▣ Ví dụ (3)



bước 5



bước 6

302

Mã Huffman

▣ Thuật toán

```

huffman (C)
begin
  n = |C|
  L = C
  for i from 1 to n-1 do // thực hiện n-1 lần hoà nhập
    x = cây có tần xuất thấp nhất trong L
    L = L - {x}
    y = cây có tần xuất thấp nhất trong L
    L = L - {y}
    left(z) = x           // z có cây con trái là x
    right(z) = y          // z có cây con phải là y
    f(z) = f(x) + f(y)
    L = L ∪ {z}
  endfor
end

```

Từ cây nhị phân mã Huffman, làm thế nào để giải mã một chuỗi nhị phân ?

303

Mã Huffman

▣ Chứng minh sự đúng đắn của thuật toán

- Chứng minh hai tính chất
 - ▣ Tính chất chọn lựa tham lam
 - ▣ Tính chất cấu trúc con tối ưu

304

Ứng dụng trong đồ thị

□ Tìm cây khung nhỏ nhất

- Thuật toán Kruskal
 - Luôn chọn cạnh ngắn nhất
- Thuật toán Prim
 - Luôn chọn cạnh ngắn nhất giữa các đỉnh thuộc cây bao phủ tối thiểu và các đỉnh không thuộc cây bao phủ tối thiểu

□ Tìm đường đi ngắn nhất

- Thuật toán Dijkstra
 - Luôn chọn cạnh ngắn nhất nối một đỉnh đã đi qua đến một đỉnh chưa đi qua

305

Cây khung nhỏ nhất (minimum spanning tree)

□ Định nghĩa

- $G=(V,E)$ là một *đồ thị* trong đó V là tập các *đỉnh* và E là tập các *cạnh* nối hai đỉnh
- Một đồ thị gọi là *vô hướng* nếu các cạnh không định hướng ($\text{cạnh}(u,v) = \text{cạnh}(v,u)$)
- Một đồ thị được gọi là *liên thông* nếu $\forall u,v \in V$ thì tồn tại đường đi nối u và v
- Một *chu trình* là một đường đi từ đỉnh v đến đỉnh v có độ dài lớn hơn không và không có cạnh nào xuất hiện hai lần
- Một *cây* là một đồ thị vô hướng, liên thông và không chứa chu trình
- Tập hợp các cây rời nhau được gọi là *rừng*
- *Cây khung* của đồ thị vô hướng $G=(V,E)$ là một cây nối tất cả các đỉnh trong V
- Cho một đồ thị vô hướng $G=(V,E)$, mỗi cạnh $e=(u,v) \in E$ có một trọng số $w(e)$. *Cây khung nhỏ nhất* T của đồ thị G là một cây khung với tổng trọng số các cạnh thuộc T là nhỏ nhất

306

Cây khung nhỏ nhất

□ Ý tưởng

- Xây dựng thuật toán tham lam xác định cây khung nhỏ nhất
 - Ở mỗi bước, cây khung nhỏ nhất được làm lớn lên bằng cách thêm vào một cạnh
 - Thuật toán thao tác trên một tập cạnh E , mà phải bảo đảm bất biến của vòng lặp như sau:
 - Trước mỗi lần lặp, A là tập con của cây khung nhỏ nhất
 - Ở mỗi bước, cạnh (u,v) được thêm vào A phải bảo đảm $A \cup \{(u,v)\}$ là tập con của cây khung nhỏ nhất
 - Cạnh (u,v) được gọi là **cạnh hợp lệ**

307

Cây khung nhỏ nhất

□ Thuật toán tham lam

```

caykhungnhonhat(G, w)
begin
  A = ∅
  while (A chưa tạo cây khung) do
    xác định cạnh (u,v) hợp lệ đối với A
    A = A ∪ {(u,v)}
  endwhile
  return (A)
end

```

- Dựa trên ý tưởng này, có hai thuật toán tìm cây khung nhỏ nhất: Kruskal và Prim
- Mỗi thuật toán sử dụng một luật riêng để xác định cạnh hợp lệ

308

Thuật toán Kruskal

- Tập A là **một rừng** (gồm nhiều cây)
- Ý tưởng
 - Từ một rừng hoà nhập các cây sao cho cuối cùng thu được cây khung nhỏ nhất
- Chọn lựa tham lam
 - Ở mỗi bước, xác định cạnh hợp lệ là cạnh (u,v) có trọng số nhỏ nhất nối hai cây trong A
- Thuật toán
 - Sắp xếp các cạnh trong E theo thứ tự giảm dần trọng lượng
 - Ban đầu, A gồm $|V|$ cây, mỗi cây chỉ chứa 1 đỉnh
 - Đối với mỗi cạnh trong E, chọn cạnh (u,v) có trọng số nhỏ nhất
 - Nếu u và v không thuộc cùng một cây trong A (để tránh tạo nên chu trình) thì (u,v) được thêm vào A và hoà nhập hai cây chứa u và v
 - Sau khi hoà nhập tất cả các cây thu được tập A chỉ còn chứa đúng 1 cây, chính là cây khung nhỏ nhất

309

Thuật toán Kruskal

□ Thuật toán

```

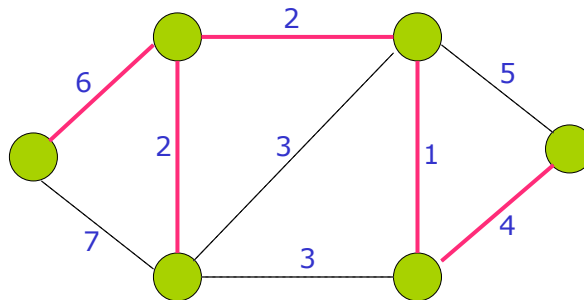
kruskal(V, E, w)
begin
  A = ∅
  sắp xếp các cạnh trong E theo thứ tự trọng số tăng dần
  for (mỗi cạnh trong E) do
    chọn cạnh (u,v) có trọng số nhỏ nhất
    if (u và v thuộc hai cây khác nhau) then
      // nối u và v không tạo chu trình
      A = A ∪ {(u,v)}
      hoà nhập hai cây chứa u và v
    endif
  endfor
  return (A)
end

```

310

Thuật toán Kruskal

▣ Ví dụ



311

Thuật toán Prim

- ▣ A là tập các cạnh tạo nên đúng **một cây**
- ▣ Ý tưởng
 - Từ một cây ban đầu chỉ là một đỉnh bất kỳ, thêm vào cây các cạnh sao cho cuối cùng thu được cây khung nhỏ nhất
- ▣ Chọn lựa tham lam
 - Ở mỗi bước, cạnh hợp lệ được thêm vào A là cạnh có trọng số thấp nhất nối cây và đỉnh không thuộc cây
- ▣ Thuật toán
 - Ban đầu, cây chỉ chứa một đỉnh r bất kỳ
 - Ở mỗi bước, chọn cạnh (u,v) - với u thuộc cây và v không thuộc cây - có trọng số nhỏ nhất
 - Thêm v vào cây và nối u và v
 - Cuối cùng, thu được cây khung nhỏ nhất

312

Thuật toán Prim

▣ Thuật toán

```

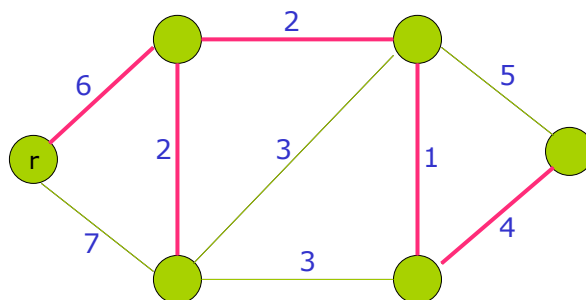
prim(V, E, w, r) // r là đỉnh bất kỳ
begin
  A = ∅
  VA = {r}
  for (mỗi đỉnh w ∈ V mà (r,w) ∈ E) do
    L = L ∪ {r,w} // danh sách các đỉnh kề r
  endfor
  while (|VA| < n) do // A chưa là cây khung
    chọn cạnh (u,v) có trọng số nhỏ nhất trong L, với u ∈ VA
    if (v ∉ VA) then // thêm đỉnh v vào cây khung
      A = A ∪ {(u,v)}
      VA = VA ∪ {v}
      for (mỗi đỉnh w ∈ V mà (v,w) ∈ E) do
        L = L ∪ {v,w} // thêm vào L các đỉnh kề v
      endfor
    endif
  endwhile
  return (A)
end

```

313

Thuật toán Prim

▣ Ví dụ



314

Cây khung nhỏ nhất

□ Độ phức tạp

- Thuật toán Kruskal $O(|E|\log(|E|))$
- Thuật toán Prim $O(|V|^2)$
- Tại sao ?

315

Đường đi ngắn nhất

- Cho đồ thị có hướng $G=(V,E)$, E tập các cung
- Các cung có trọng số không âm
- Gọi s là đỉnh xuất phát
- Xác định đường đi ngắn nhất từ đỉnh s đến mỗi đỉnh còn lại

□ Thuật toán Dijkstra

316

Thuật toán Dijkstra

□ Ý tưởng

- Gọi S là tập các đỉnh đã xác định được đường đi ngắn nhất từ s đến
 - mỗi $u \in S$ ta có $d[u]$ là độ dài đường đi ngắn nhất từ s đến u
 - mỗi $u \notin S$ ta có $d[u]$ là độ dài đường đi ngắn nhất từ s đến u qua các đỉnh thuộc S : $d[u] = \min(d[x] + w[x, u], \forall x \in S)$
 - xuất phát, $S = \{s\}$
- Chọn lựa tham lam
 - ở mỗi bước, thêm vào S đỉnh v sao cho đường đi từ s đến v qua các đỉnh thuộc S là ngắn nhất
 - mỗi khi thêm vào S đỉnh v thì phải tính lại độ dài đường đi ngắn nhất từ s đến các đỉnh còn lại trong V qua các đỉnh thuộc S
- Cuối cùng, khi $S = V$ thì dừng lại

317

Thuật toán Dijkstra

□ Thuật toán (1)

```

dijkstra(V, E, w[1..n,1..n])
begin
  S = {s}
  for (mỗi u ∈ V) do // khởi tạo đường đi ngắn nhất từ s
    d[u] = w[s,u]
  endfor
  for i from 1 to n-1 do // lặp n-1 lần, để thêm n-1 đỉnh vào S
    chọn v ∈ V-S mà có d[v] nhỏ nhất
    S = S ∪ {v}
    for (mỗi u ∈ V-S) do // tính lại đường đi ngắn nhất từ s đến u
      d[u] = min(d[u], d[v] + w[v,u])
    endfor
  endfor
  return (d)
end

```

Thuật toán Dijkstra

□ Thuật toán (2)

- Thuật toán mới chỉ tính độ dài của đường đi ngắn nhất
- Cần lưu lại đường đi ngắn nhất
 - Dùng mảng p , với $p[v]$ chứa đỉnh đứng trước v trong đường đi ngắn nhất từ s đến v qua các đỉnh trong S

319

Bài tập (1)

□ Bài 1

- Cho tập hợp A gồm n số nguyên, tìm tập hợp con S của A thoả mãn:
 - (i) có đúng m phần tử ($m \leq n$)
 - (ii) tổng các phần tử của S là lớn nhất
- 1. Xây dựng thuật toán tham lam xác định S
- 2. Chứng minh thuật toán tối ưu

□ Bài 2

- Một người cắt tóc phục vụ n khách hàng. Mỗi khách hàng cần một thời gian phục vụ t_i khác nhau. Mỗi thời điểm người cắt tóc chỉ có thể phục vụ một khách hàng.
- 1. Đề xuất thuật toán vét cạn
- 2. Xây dựng thuật toán tham lam lập lịch phục vụ các khách hàng sao cho tổng thời gian chờ và được phục vụ của các khách hàng là nhỏ nhất
- 3. Chứng minh thuật toán tối ưu
- 4. So sánh độ phức tạp của thuật toán tham lam và thuật toán vét cạn

320

Bài tập (2)

□ Bài 3

- Có n công việc, mỗi công việc cần 1 đơn vị thời gian để hoàn thành. Nếu mỗi công việc i bắt đầu trước hoặc tại thời điểm d_i thì sẽ mang lại lợi ích g_i .

Xây dựng thuật toán tham lam lập lịch các công việc sao cho tổng lợi ích thu được là lớn nhất (lưu ý, phụ thuộc vào các thời điểm d_i không nhất thiết tất cả các công việc đều được lập lịch)

321

Giải thích 1

□ Chứng minh

- Định lý: một cây nhị phân đầy đủ có n nút lá thì có $n-1$ nút trong
- Quy nạp
 - **Bước cơ sở:** đúng khi $n = 1$, có 0 nút trong
 - **Giả thiết:** định lý đúng với cây nhị phân đầy đủ có số nút lá nhỏ hơn n
 - **Suy dẫn:** Giả sử cây nhị phân T gồm nút gốc r và các cây con T_1, T_2 có n nút lá. Giả sử T_1 (tương ứng T_2) có n_1 (n_2) nút lá. Vậy $n = n_1 + n_2$.
Số nút trong của cây T bằng số nút trong của cây T_1 , cây T_2 và thêm nút gốc r .
Áp dụng giả thiết quy nạp cho cây T_1 và cây T_2 , số nút trong của T_1 (tương ứng T_2) là n_1-1 (n_2-1).
Vậy số nút trong của cây T là:
$$(n_1-1) + (n_2-1) + 1 = n_1 + n_2 - 1 = n - 1$$



322

Quay lui (8)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Quay lui (backtracking)

- *Tìm kiếm vết cạn* trong một không gian trạng thái của bài toán
 - Các giải pháp của bài toán được biểu diễn bởi một không gian trạng thái (cụ thể là một cây)
 - Tìm kiếm giải pháp = tìm kiếm vết cạn trên không gian trạng thái
- Thường được sử dụng để giải quyết các bài toán yêu cầu tìm kiếm các phần tử của một tập hợp thoả mãn một số *ràng buộc*
- Nhiều bài toán được giải quyết bởi thuật toán quay lui có dạng:

« *Tìm tập con S của $A_1 \times A_2 \times \dots \times A_n$ (A_k là một tập hợp) sao cho mỗi phần tử $s = (s_1, s_2, \dots, s_n) \in S$ thoả mãn ràng buộc nào đó* »
- Ví dụ
 - Tìm tất cả các hoán vị của $\{1, 2, \dots, n\}$
 $A_k = \{1, 2, \dots, n\}$ với $\forall k$
 $s_i \neq s_k$ với $\forall i \neq k$

324

Quay lui

□ Ý tưởng

- Giải pháp được xây dựng từng thành phần ở mỗi bước
- *Tìm kiếm vết cạn* tất cả các giải pháp có thể trên cây không gian trạng thái
 - Mất nhiều thời gian thực thi
- *Tỉa bớt* các thành phần không đưa đến giải pháp
 - Chỉ những giải pháp từng phần có triển vọng được sử dụng
 - Giải pháp từng phần có triển vọng nếu nó có thể dẫn đến giải pháp cuối cùng, nếu không thì gọi là giải pháp không có triển vọng
 - Những giải pháp từng phần không có triển vọng sẽ bị loại bỏ
- Nếu tất cả các giá trị của một thành phần không dẫn đến một giải pháp từng phần có triển vọng thì quay lui thành phần trước và thử giá trị khác

325

Quay lui

□ Giải pháp quay lui xây dựng không gian trạng thái dưới dạng cây

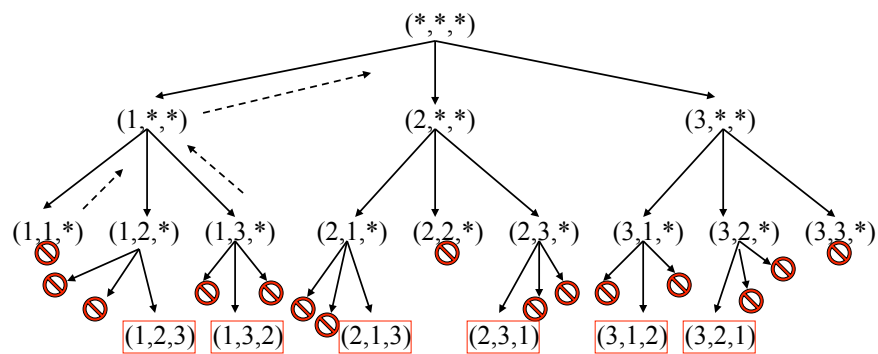
- Nút gốc tương ứng với trạng thái đầu (trước khi việc tìm kiếm giải pháp bắt đầu)
- Mỗi nút trong tương ứng với một giải pháp từng phần có triển vọng
- Các nút lá tương ứng với hoặc giải pháp từng phần không có triển vọng hoặc giải pháp cuối cùng

326

Quay lui

▣ Ví dụ

■ Cây không gian trạng thái



327

Quay lui

▣ Tìm kiếm giải pháp

- Tìm kiếm theo chiều sâu trước trên cây không gian trạng thái
- Nếu tìm kiếm theo chiều sâu gặp một nút lá
 - ▣ thì kiểm tra giải pháp hiện tại có thoả mãn các ràng buộc hay không
 - ▣ có thể đưa thêm các điều kiện để kiểm tra một giải pháp có tối ưu không

328

Quay lui

▣ Các bước thiết kế thuật toán quay lui

1. Chọn cách *biểu diễn giải pháp*
2. Xây dựng các *tập* A_1, A_2, \dots, A_n và xếp thứ tự để các phần tử của chúng được xử lý
3. Xây dựng các *điều kiện* từ ràng buộc của bài toán để xác định một giải pháp từng phần có là triển vọng không, gọi là *điều kiện tiếp tục*
4. Chọn *tiêu chí* để xác định một giải pháp từng phần có là *giải pháp cuối cùng* không

329

Quay lui

▣ Ví dụ: hoán vị của $\{1, 2, \dots, n\}$

- *biểu diễn giải pháp*
 - ▣ mỗi hoán vị là một véc-tơ $s = (s_1, s_2, \dots, s_n)$, $s_i \neq s_j$ với $\forall i \neq j$
- *tập* A_1, A_2, \dots, A_n và *thứ tự* các phần tử
 - ▣ $A_k = \{1, 2, \dots, n\}$, mọi k
 - ▣ các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - ▣ mỗi giải pháp từng phần $s = (s_1, s_2, \dots, s_k)$, $s_i \neq s_k$ với $\forall i \neq k$
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - ▣ $k = n$

330

Quay lui

▣ Thuật toán quay lui tổng quát: đệ quy

```

quaylui-dequy (k)
begin
  if (s=(s1, s2, ..., sk-1) là giải pháp) then xuly(s)
  else
    for j from 1 to |Ak| do // thử tất cả các giá trị có thể
      sk = ajk // Ak = {a1k, a2k, ...}
      if ((s1, s2, ..., sk) là có triển vọng) then
        quaylui-dequy (k+1) // xây dựng thành phần tiếp theo
      endif
    endfor
  endif
end

```

331

Quay lui

▣ Thuật toán quay lui tổng quát: lặp

```

quaylui-lap(A1, A2, ..., An)
begin
  k=1; ik=0
  while (k > 0) do
    ik=ik+1
    v=false
    while (v=false and ik≤|Ak|) do
      sk=aikk
      if ((s1, ..., sk) là có triển vọng) then v=true
      else ik=ik+1 endif // thử giá trị tiếp theo
    endwhile
    if (v=true) then
      if (s=(s1, ..., sk) là giải pháp cuối cùng) then xuly(s)
      else k=k+1 // xây dựng thành phần tiếp theo
      ik=0
    endif
    else k=k-1 endif // quay lui lại thành phần trước đó
  endwhile
end

```

332

Mối số ứng dụng

- Hoán vị
- Chuỗi nhị phân
- Tập con
- Xếp n con hậu
- Tìm đường đi
- Xếp ba lô 0-1
- Mê cung

333

Hoán vị

- Bài toán
 - Tìm tất cả các hoán vị của $\{1, 2, \dots, n\}$
- Các bước thiết kế
 - *biểu diễn giải pháp*
 - mỗi hoán vị là một véc-tơ $s = (s_1, s_2, \dots, s_n)$, $s_i \neq s_j$ với $\forall i \neq j$
 - *tập A_1, A_2, \dots, A_n và thứ tự các phần tử*
 - $A_k = \{1, 2, \dots, n\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
 - *điều kiện tiếp tục*
 - mỗi giải pháp từng phần $s = (s_1, s_2, \dots, s_k)$, $s_i \neq s_k$ với $\forall i \neq k$
 - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - $k = n$

334

Hoán vị

Thuật toán

```

hoanvi (k)
begin
  if (k=n+1) then print(s[1..n])
  else
    for i from 1 to n do
      s[k]=i
      if (trienvong(s[1..k])) then
        hoanvi (k+1)
      endif
    endfor
  endif
end

```

```

trienvong(s[1..k])
begin
  for i from 1 to k-1 do
    if (s[k]=s[i]) then
      return false
    endif
  endfor
  return true
end

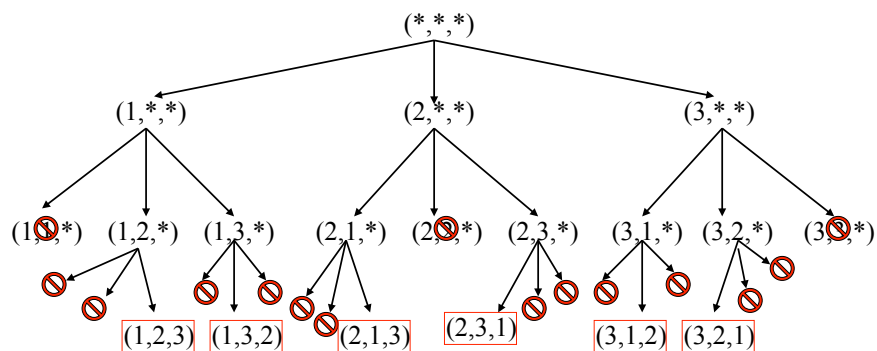
```

sử dụng: hoanvi(1)

335

Hoán vị

Minh họa: n=3



336

Chuỗi nhị phân

□ Bài toán

- Tìm tất cả các chuỗi nhị phân có độ dài n

□ Các bước thiết kế

- *biểu diễn giải pháp*
 - mỗi chuỗi nhị phân là một véc-tơ $s=(s_1, s_2, \dots, s_n)$
- *tập A_1, A_2, \dots, A_n và thứ tự các phần tử*
 - $A_k = \{0, 1\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - mọi giải pháp từng phần đều có triển vọng
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - $k = n$

337

Chuỗi nhị phân

□ Thuật toán

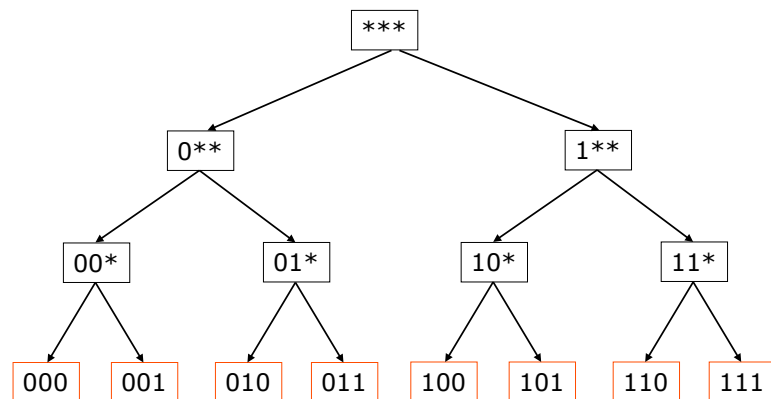
```

chuoinhiphan (k)
begin
  if (k=n+1) then print(s[1..n])
  else
    for i from 0 to 1 do
      s[k]=i
      chuoinhiphan (k+1)
    endfor
  endif
end
  
```

338

Chuỗi nhị phân

▣ Minh họa: $n = 3$



339

Chuỗi nhị phân

▣ Phân tích thuật toán

- $C(n)$ thời gian thực thì $chuoinhiphan(n)$

- Vậy

$$C(n) = \begin{cases} c & \text{if } n = 0 \\ 2C(n-1) + d & \text{else} \end{cases}$$

- Bằng phương pháp thay thế, ta có

$$C(n) = 2^n(c+d) - d$$

- Vậy $C(n) = O(2^n)$

- Thuật toán tối ưu, vì có 2^n chuỗi nhị phân có độ dài bằng n

- Không cần bước xóa bớt các giải pháp thành phần

340

Tập con

□ Bài toán

- Cho tập hợp $A = \{a_1, a_2, \dots, a_n\}$, hãy sinh ra tất cả các tập con của A

□ Các bước thiết kế

- *biểu diễn giải pháp*
 - mỗi tập hợp con được biểu diễn bởi một véc-tơ $s = (s_1, s_2, \dots, s_n)$, nếu $s_i = 1$ thì a_i thuộc s , $s_i = 0$ thì ngược lại
- *tập A_1, A_2, \dots, A_n và thứ tự các phần tử*
 - $A_k = \{0, 1\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - giải pháp từng phần $s = (s_1, s_2, \dots, s_k)$ có ít hơn n phần tử, hay $k < n$
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - $k = n$

341

Tập con

□ Thuật toán

```

tapcon (k)
begin
  if (k-1=n) then
    print (s[1..k-1])
  else
    s[k]=0; tapcon (k+1)
    s[k]=1; tapcon (k+1)
  endif
end

```

- Tại sao không có hàm *trienvong* kiểm tra điều kiện tiếp tục ?
- Chính sửa thuật toán để tạo ra tất cả các tập hợp con của A có đúng m phần tử ($m < n$)

342

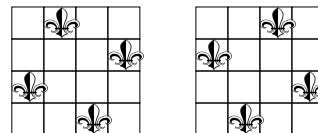
Xếp n con hậu

□ Bài toán

- Tìm tất cả các khả năng xếp n con hậu trên một bàn cờ có $n \times n$ ô sao cho các con hậu không tấn công nhau, nghĩa là
 - mỗi *hàng* chỉ chứa một con hậu
 - mỗi *cột* chỉ chứa một con hậu
 - mỗi *đường chéo* chỉ chứa một con hậu

■ Ví dụ

- $n=3$: không tồn tại giải pháp
- $n=4$: có hai giải pháp
- $n=8$: có 92 giải pháp



343

Xếp n con hậu

□ Các bước thiết kế

- *biểu diễn giải pháp*
 - giả sử con hậu k được đặt trên hàng k, như thế đối với mỗi con hậu chỉ cần mô tả cột chứa nó. Khi đó giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \dots, s_n)$ với $s_k = \text{cột mà con hậu k được đặt trên đó}$
- tập A_1, A_2, \dots, A_n và thứ tự các phần tử
 - $A_k = \{1, \dots, n\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - giải pháp từng phần $s=(s_1, s_2, \dots, s_k)$ phải thỏa mãn ràng buộc bài toán (mỗi hàng/cột/đường chéo chỉ chứa đúng một con hậu)
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - $k = n$

344

Xếp n con hậu

□ Điều kiện tiếp tục (1)

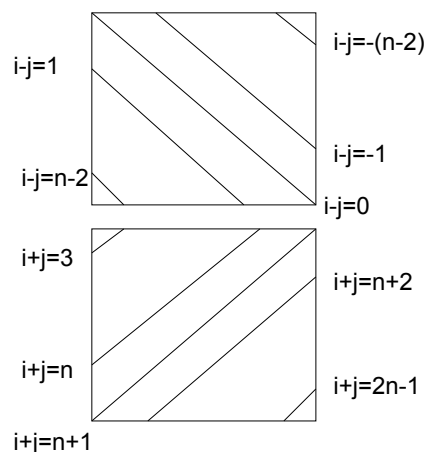
- giải pháp từng phần $s=(s_1, s_2, \dots, s_k)$ phải thoả mãn ràng buộc bài toán (mỗi hàng/cột/đường chéo chỉ chứa đúng một con hậu)
- mỗi hàng chứa đúng một con hậu: điều này luôn đúng do cách biểu diễn giải pháp
- mỗi cột chứa đúng một con hậu: $s_i \neq s_k$ với mọi $i \neq k$. Chỉ cần kiểm tra $s_k \neq s_i$ với mọi $i \leq k-1$
- Mỗi đường chéo chỉ chứa một con hậu: $|j-i| \neq |s_j-s_i|$ với mọi $i \neq j$. Chỉ cần kiểm tra $|k-i| \neq |s_k-s_i|$ với mọi $i \leq k-1$
 - Tại sao ?

345

Xếp n con hậu

□ Điều kiện tiếp tục (2)

- Hai con hậu i và j ở trên cùng một đường chéo nếu:
 $i-s_i = j-s_j$ hay $j-i = s_j-s_i$
 hoặc
 $i+s_i = j+s_j$ hay $j-i = s_i-s_j$
 nghĩa là: $|j-i| = |s_j-s_i|$



346

Xếp n con hậu

Thuật toán

```

xephau (k)
begin
  if (k=n+1) then
    print(s[1..n])
  else
    for i from 1 to n do
      s[k]=i
      if (trienvong(s[1..k])) then
        xephau(k+1)
      endif
    endfor
  endif
end
  
```

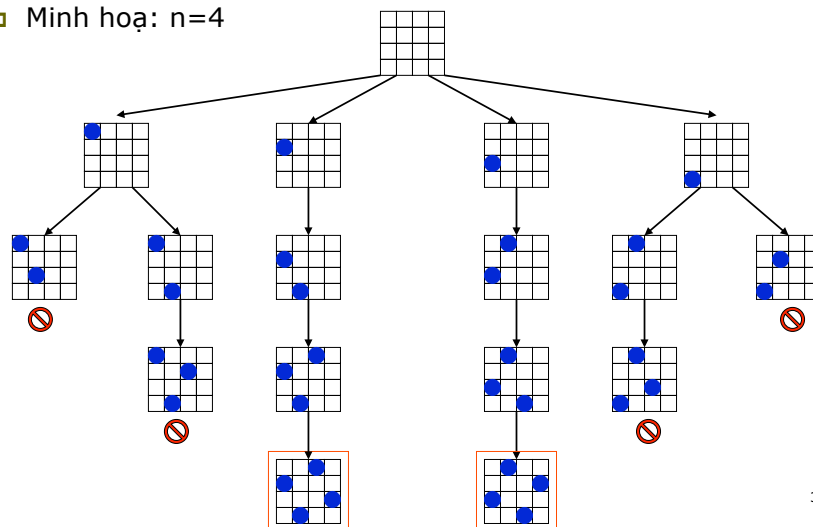
```

trienvong(s[1..k])
begin
  for i from 1 to k-1 do
    if (s[k]=s[i] or |i-k|=|s[i]-s[k]|)
      then return false
    endif
  endfor
  return true
end
  
```

347

Xếp n con hậu

Minh họa: n=4



348

Xếp n con hậu

□ Nhận xét

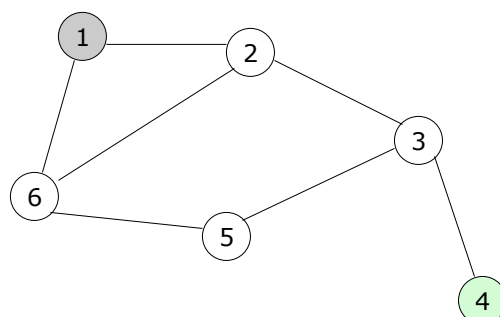
- Với $n=4$
 - Nếu sử dụng kỹ thuật *tia bắt* chỉ có 2 giải pháp cuối cùng được xét
 - Nếu không sử dụng kỹ thuật *tia bắt*, nghĩa là tìm tất cả các khả năng đặt n con hậu trên bàn cờ, sau đó kiểm tra xem mỗi khả năng có hợp lệ không.
Có $C_n^{n^2}$ khả năng
Với $n = 4$, có 1820 khả năng
- Như vậy, kỹ thuật *tia bắt* của thuật toán quay lui tiết kiệm rất lớn thời gian thực thi

349

Tìm đường đi

□ Bài toán

- Cho tập hợp n thành phố, có một mạng lưới giao thông nối các thành phố này. Tìm tất cả các đường đi nối hai thành phố cho trước sao cho không đi qua một thành phố nào hai lần



Đi từ 1 đến 4

1→2→3→4

1→2→6→5→3→4

1→6→2→3→4

1→6→5→3→4

350

Tìm đường đi

□ Phân tích bài toán

- Giả sử mạng lưới giao thông nối các thành phố được mô tả bởi ma trận $C[1..n, 1..n]$

$$C[i, j] = \begin{cases} 0 & \text{nếu không tồn tại đường đi trực tiếp giữa } i \text{ và } j \\ 1 & \text{nếu không tồn tại đường đi trực tiếp giữa } i \text{ và } j \end{cases}$$

- Tìm tất cả các đường đi $s = (s_1, s_2, \dots, s_m)$, s_k trong $\{1, \dots, n\}$ chỉ ra thành phố đi đến ở bước k sao cho
 - s_1 là thành phố xuất phát
 - s_m là thành phố đích
 - $C[s_{i-1}, s_i] = 1$ (tồn tại đường đi nối trực tiếp giữa hai thành phố s_i và s_{i+1})
 - $s_i \neq s_j$ với mọi $i \neq j$ (một thành phố không được đi qua hai lần)

351

Tìm đường đi

□ Các bước thiết kế

- *biểu diễn giải pháp*
 - giải pháp được biểu diễn bởi véc-tơ $s = (s_1, s_2, \dots, s_m)$ với s_k thành phố đi đến ở bước k
- *tập A_1, A_2, \dots, A_n và thứ tự các phần tử*
 - $A_k = \{1, \dots, n\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - giải pháp từng phần $s = (s_1, s_2, \dots, s_k)$ phải thoả mãn:
 - $s_i \neq s_k$ với mọi i trong $\{1, \dots, k-1\}$
 - $C[s_{k-1}, s_k] = 1$
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - s_k = thành phố đích

352

Tìm đường đi

Thuật toán

```

duongdi (k)
begin
  if (s[k-1]=thành phố đích) then
    print(s[1..k-1])
  else
    for j from 1 to n do
      s[k]=j
      if (trienvong(s[1..k])) then
        duongdi (k+1)
      endif
    endfor
  endif
end

```

s_1 = thành phố xuất phát
duongdi(2)

```

trienvong(s[1..k])
begin
  if (C[s[k-1],s[k]] = 0) then
    return false
  endif
  for i from 1 to k-1 do
    if (s[i]=s[k]) then
      return false
    endif
  endfor
  return true
end

```

353

Xếp ba lô 0-1

Bài toán

- Có n đồ vật có trọng lượng w_1, \dots, w_n và giá trị tương ứng v_1, \dots, v_n . Xếp các đồ vật vào ba lô có sức chứa W sao cho tổng giá trị lớn nhất

Thuật toán quay lui liệt kê tất cả các giải pháp có thể hoặc giải pháp đầu tiên tìm thấy

Bài toán xếp ba lô 0-1 cần một giải pháp tối ưu

- Khi tìm được một giải pháp, thì so sánh với giải pháp trước đó để xác định giải pháp tốt hơn
- Cuối cùng, tìm được **giải pháp tốt nhất**

354

Xếp ba lô 0-1

□ Các bước thiết kế

■ biểu diễn giải pháp

- giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \dots, s_n)$ với nếu $s_i = 1$ thì đồ vật i được chọn, $s_i = 0$ ngược lại thì không

■ tập A_1, A_2, \dots, A_n và thứ tự các phần tử

- $A_k = \{1, 0\}$, mọi k
- các phần tử sẽ được xử lý tăng dần

■ điều kiện tiếp tục

■ tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng

- $k = n$

$$\sum_{i=1}^k s_i w_i \leq W$$

- So sánh giải pháp với giải pháp trước đó, lưu lại giải pháp có tổng giá trị lớn nhất

355

Xếp ba lô 0-1

□ Thuật toán

giatri-totnhat được khởi gán bằng 0

```

xepbalo (k)
begin
  if (k-1 = n) then
    if (  $\sum_{i=1}^n s_i w_i \leq W$  ) then
      giatri =  $\sum_{i=1}^n s_i v_i$ 
      if (giatri > giatri-totnhat) then
        giatri-totnhat=giatri
        giaphap-totnhat={ $s_1, s_2, \dots, s_n$ }
      endif
    endif
  else
    s[k] = 0; xepbalo (k+1)
    s[k] = 1; xepbalo (k+1)
  endif
end

```

Chọn giải pháp tốt hơn

356

Xếp ba lô 0-1

□ Cải tiến thuật toán (1)

- Tỉa bớt các lời gọi đệ quy không bao giờ cho giải pháp
- Chỉnh lại *điều kiện tiếp tục*
- *điều kiện tiếp tục*
 - giải pháp từng phần $s=(s_1, s_2, \dots, s_k)$ phải thoả mãn:

$$\sum_{i=1}^k s_i w_i \leq W$$

357

Xếp ba lô 0-1

□ Cải tiến thuật toán (2)

Tỉa bớt
Có thể thay
bằng hàm
triển vọng

```
xepbalo (k)
begin
  if (k-1 = n) then
    giatri =  $\sum_{i=1}^n s_i v_i$ 
    if (giatri > giatri-totnhat) then
      giatri-totnhat=giatri
      giaphap-totnhat={s1,s2,...,sn}
    endif
  else
    s[k] = 0; xepbalo (k+1)
    if (  $\sum_{i=1}^k s_i w_i \leq W$  ) then
      s[k] = 1; xepbalo (k+1)
    endif
  endif
end
```

358

Xếp ba lô 0-1

Thuật toán đơn giản hơn

```

xepbalo (k, W)
begin
  if (k-1 = n) then
    giatri =  $\sum_{i=1}^n s_i v_i$ 
    if (giatri > giatri-totnhat) then
      giatri-totnhat = giatri
      giaphap-totnhat = {s1, s2, ..., sn}
    endif
  else
    s[k] = 0; xepbalo (k+1, W)
    if (W ≥ wk) then
      s[k] = 1; xepbalo (k+1, W-wk)
    endif
  endif
end

```

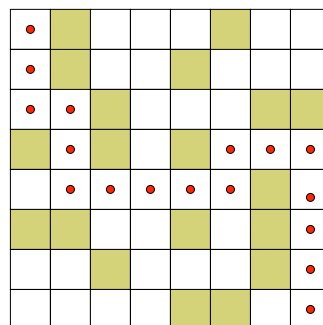
359

W được khởi gán là sức chứa tối đa của ba lô

Mê cung

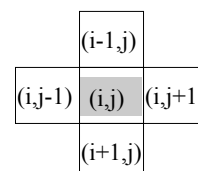
Bài toán

- Giả sử một mê cung được định nghĩa là một lưới có $n \times n$ ô. Tìm đường đi trong mê cung xuất phát từ ô (1,1) đến ô (n,n)



Chỉ có thể đi qua những ô rỗng

Từ một ô (i,j) có thể đến được một trong các ô:
(i-1,j), (i+1,j), (i,j-1), (i,j+1)



360

Mê cung

□ Phân tích bài toán

- Dùng ma trận $M[1..n, 1..n]$ để lưu trữ mê cung sao cho

$$M[i,j] = \begin{cases} 0 & \text{nếu ô (i,j) rỗng} \\ 1 & \text{nếu ô (i,j) đặc} \end{cases}$$

- Tìm một đường đi $s=(s_1, s_2, \dots, s_m)$, s_k trong $\{1, \dots, n\} \times \{1, \dots, n\}$ chỉ ra chỉ số tương ứng ô đi đến ở bước k sao cho
 - s_1 là ô xuất phát $(1,1)$
 - s_m là ô đích (n,n)
 - $s_k \neq s_q$ với mọi $k \neq q$ (mỗi ô chỉ đi qua nhiều nhất một lần)
 - $M(s_k) = 0$ (ô được đi đến phải rỗng)
 - s_{k-1} và s_k là các ô kề nhau

361

Mê cung

□ Các bước thiết kế

- *biểu diễn giải pháp*
 - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \dots, s_m)$ với s_k ô đi đến ở bước k
- *tập A_1, A_2, \dots, A_n và thứ tự các phần tử*
 - $A_k = \{1, \dots, n\} \times \{1, \dots, n\}$, mọi k
 - các phần tử sẽ được xử lý tăng dần
- *điều kiện tiếp tục*
 - giải pháp từng phần $s=(s_1, s_2, \dots, s_k)$ phải thoả mãn:
 - $s_k \neq s_q$ với mọi q trong $\{1, \dots, k-1\}$
 - $M(s_k) = 0$
 - s_{k-1} và s_k là các ô kề nhau
- *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
 - s_k là ô đích (n,n)

362

Mê cung

Thuật toán (1)

```

mecung (k)
begin
  if (s[k-1]=(n,n)) then print (s[1..k-1])
  else
    s[k].i=s[k-1].i-1; s[k].j=s[k-1].j      // đi lên
    if (trienvong(s[1..k])) then mecung(k+1) endif
    s[k].i=s[k-1].i+1; s[k].j=s[k-1].j      // đi xuống
    if (trienvong(s[1..k])) then mecung(k+1) endif
    s[k].i=s[k-1].i; s[k].j=s[k-1].j-1      // qua trái
    if (trienvong(s[1..k])) then mecung(k+1) endif
    s[k].i=s[k-1].i-1; s[k].j=s[k-1].j+1    // qua phải
    if (trienvong(s[1..k])) then mecung(k+1) endif
  endif
end

```

363

Mê cung

Thuật toán (2)

```

trienvong (s[1..k])
begin
  if (s[k].i<1 or s[k].i>n or s[k].j<1 or s[k].j>n) then
    return false // ô ngoài mê cung
  endif
  if (M[s[k].i,s[k].j]=1) then return false endif
  for q from 1 to k-1 do
    if (s[k].i=s[q].i and s[k].j=s[q].j) then return false endif
  endfor
  return true
end

```

Sử dụng: $s[1] = (1,1)$
 mecung(2)

364

Bài tập

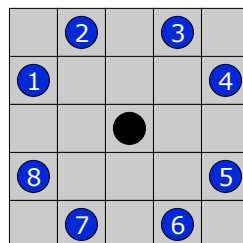
- Tổng tập con
 - Cho tập S gồm n số nguyên, tìm tất cả các tập con của S sao cho tổng các phần tử của nó bằng đúng W ($W > 0$)
- Chu trình Hamilton
 - Chu trình Hamilton là một chu trình trong đồ thị vô hướng đi qua mỗi đỉnh đúng một lần và quay lại đỉnh xuất phát. Xác định tất cả các chu trình Hamilton trong một đồ thị vô hướng
- Tô màu bản đồ
 - Cho bản đồ gồm n nước. Hãy tìm cách tô màu bản đồ sử dụng $m \geq 4$ màu sao cho hai nước láng giềng bất kỳ có màu khác nhau

365

Bài tập

- Ngựa đi tuần
 - Xuất phát từ một ô bất kỳ trên bàn cờ 8×8 ô, tìm cách đi con ngựa qua tất cả các ô trên bàn cờ

Luật di chuyển của con ngựa



Từ ô hiện hành có thể di chuyển đến 8 ô khác nhau

366