Thuật toán nâng cao

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 thuật toán
- Các kỹ thuật thiết kế thuật toán
- Vận dụng giải quyết các bài toán cụ thể

Nội dung

Đánh giá

- Bài tập lớn 1
  - Tóm tắt bài báo khoa học
- Bài tập lớn 2
  - Tìm hiểu và trình bày một chủ đề về thuật toán
- Thi kết thúc môn học

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 Comlexity, 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.

Bài tập lớn

- Mỗi học viên chọn trình bày **một trong các chủ đề** sau
  - Kỹ thuật nhánh và cận (branch and bound)
  - Kỹ thuật biến đổi Fourier (Fourier transform)
  - Thuật toán di truyền (genetic algorithm)
- Yêu cầu
  - Tìm hiểu và trình bày về chủ đề
  - Minh hoạ và giải thích phương pháp bởi ít nhất 2 ví dụ cụ thể
  - Đánh giá và phân tích giải pháp các ví dụ
  - Học viên tự tìm kiếm các tài liệu liên quan đến chủ đề
- Kết quả
  - Báo cáo từ 10 đến 20 trang
  - Báo cáo phải **trích dẫn rỏ ràng** các tài liệu tham khảo

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 $ax^2 + bx + c = 0$: rỏ 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

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

9

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
  - …

10

Giới thiệu

- Các tính chất của thuật toán
  - Tính tổng quát
  - Tính hữu hạn
  - Tính không nhập nhằng
  - Tính hiệu quả

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

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:	2	1	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

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 ?

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 xuống 1
Bước 3: if (x ∈[-2,2]) then
             goto Bước 2
```

  - Thuật toán có nhập nhằng ?

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*gt(n-1))
  endif
end
```

  - Thuật toán nào hiệu quả hơn ?

Giới thiệu

- Đặc tả thuật toán (1)
  - Có nhiều cách đặc tả thuật toán
    - Không hình thức
      - Ngôn ngữ tự nhiên
    - Nữa hình thức
      - Kêt 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, …

Giới thiệu

- Đặc tả thuật toán (2)
  - Chọn phương pháp đặc tả nữa 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
```

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?

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 xuất lấy các loại đinh ngang nhau, phải mở 8 ngăn kéo

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

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

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 đinh cần lấy
      - Nếu không, chúng ta loại bỏ ngăn kéo vừa mở, thực hiện mở ngăn kéo khác một cách ngẫu nhiên

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

21

Giới thiệu

- Ví dụ 1
  - Giải pháp tiền xử lý
    - Chúng ta sắp xếp các hộp đinh 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 đinh

22

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân hai số nguyên
  - Thuật toán nhân Bắc Mỹ

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

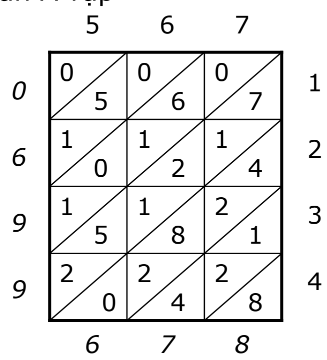

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân Anh

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

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân Ả-rập



  - So sánh độ hiệu quả các thuật toán ?

Giới thiệu

- Ví dụ 2
  - Đếm số phép toán nhân và cộng được thực hiện
    - Thuật toán nhân Bắc Mỹ và thuận toán nhân Anh đều sử dụng 12 phép nhân và 15 phép cộng
    - Thuật toán Ả-rập sử dụng 12 phép nhân và 20 phép cộng
    - Số phép toán phụ thuộc vào số chữ số của mỗi số nguyên
      - Số phép nhân bằng m*n với m và n lần lượt là số chữ số của mỗi số nguyên

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

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân « chia để trị »
    - Số chữ số của hai số nguyên phải bằng nhau và phải bằng luỹ 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 nữa trái và phải của số a, tương tự b_L và b_R là hai nữa trái và phải của b

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân « chia để trị »

$$
\begin{array}{rccc}
 & & a_L & a_R \\
 & * & b_L & b_R \\
\hline
 & & a_L b_R & a_R b_R \\
+ & a_L b_L & a_R b_L & \\
\hline
\end{array}
$$

$$a_L b_L + (a_L b_R + a_R b_L) + a_R b_R$$

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

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

29

Giới thiệu

- Ví dụ 2
  - Thuật toán nhân « chia để trị »
    - Giảm số phép nhân a và b bởi các nữa của a và b từ 4 xuống còn 3

      Đặt: $p = a_L*b_L$
      $q = a_R*b_R$
      $r = (a_L + a_R)*(b_L + a_R)$

      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

30

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
		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
		170

31

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

```
if n = 0 then
        y = 1
else
        y = x
endif
for i from 1 to n do
        y = y * x
endfor
```

32

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 $x^n = x^{n-1} * x$
    - Chẳng hạn
      - $x^{35} = x^{34} * x$
      - $x^{34} = x^{17} * x^{17}$
      - $x^{17} = x^{16} * x$
      - $x^{16} = x^8 * x^8$
      - $x^8 = x^4 * x^4$
      - $x^4 = x^2 * x^2$
      - $x^2 = x * x$

Giới thiệu

- Ví dụ 3: tính x^n
  - Thuật toán nhị phân
    1. Biểu diễn n bởi dãy nhị phân
    2. Thay mỗi bít nhị phân
       « 1 » bởi các kí hiệu SX
       « 0 » bởi các kí hiệu X
    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

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}...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
```

35

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

36

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 bít bên 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:

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

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

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 $s > 1$
  - 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}

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

Chứng minh sự đúng đắn (2)

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

Phân tích thuật toán

- Kiểm tra sự đúng đắn
  - Chỉ ra rằng thuật toán cho kết quả như mong đợi sau một số bước thực hiện
- Đánh giá hiệu quả
  - Đánh giá nguồn tài nguyên thuật toán sử dụng của máy tính
    - Thời gian
    - Bộ nhớ

Kiểm tra tính đúng đắn

- Thực nghiệm
  - Kiểm thử (testing)
    - Thực thi thuật toán trên tập các dữ liệu vào và quan sát kết quả
- Lý thuyết
  - Chứng minh sự đúng đắn (correctness proof)
    - Chứng minh rằng thuật toán cho kết quả đúng với mọi dữ liệu vào

Ưu nhược điểm

	Thực nghiệm	**Lý thuyết**
Ưu điểm	-Đơn giản hơn -Dễ thực hiện	-Bảo đảm tính đúng đắn
Nhược điểm	-Không bảo đảm hoàn toàn tính đúng đắn	-Khó thực hiện -Không thể áp dụng cho các thuật toán phức tạp

Chứng minh sự đúng đắn

- Một vài khái niệm
  - Tiền điều kiện và hậu điều kiện
  - Trạng thái của thuật toán
  - Các xác nhận
  - Chú thích thuật toán

Tiền điều kiện và hậu điều kiện

- Tiền điều kiện (preconditions)
  - Các tính chất mà dữ liệu vào phải thoả mãn
- Hậu điều kiện (postconditions)
  - Các tính chất mà kết quả của thuật toán phải thoả mãn
- Ví dụ
  - Tìm giá trị *m* nhỏ nhất trong mảng không rỗng x[1..n]

    preconditions: $n \geq 1$

    postconditions: $m = \min(x[i] \mid 1 \leq i \leq n)$

Trạng thái của thuật toán

- Trạng thái của thuật toán
  - là tập các *giá trị tương ứng với tất cả các biến được sử dụng* trong thuật toán
- Trong quá trình thực thi trạng thái thuật toán thay đổi
- Thuật toán là đúng nếu cuối cùng trạng thái của nó thoả mãn hậu điều kiện

Các xác nhận

- Xác nhận (assertion)
  - là một câu lệnh mô tả các ràng buộc trên trạng thái của thuật toán
- Ví dụ

```
{x > 0}
x = x + y
{x > y}
```

- Các xác nhận được sử dụng
  - Chứng minh sự đúng đắn
  - Chú thích thuật toán
  - Viết tài liệu

Chú thích thuật toán

- Sử dụng các xác nhận để chú thích thuật toán
- Ví dụ

```
min (x[1..n])
begin
    {n ≥ 1}
    m = x[1]
    {m = x[1]}
    for i from 2 to n do
    {2 ≤ i ≤ n}
        if (m > x[i]) then
            m = x[i]
        endif
        {m = min(x[1], x[2], …, x[i])}
    endfor
    return (m)
end
```


Chứng minh sự đứng đắn

- Chứng minh đúng đắn một phần (partial correctness)
  - Chứng minh rằng khi dữ liệu vào thoả mãn tiền điều kiện thì kết quả thuật toán sẽ thoả mãn hậu điều kiện
- Chứng minh đúng đắn toàn phần (total correctness)
  - Chứng minh rằng thuật toán đúng đắn một phần và thuật toán dừng
- Các bước trung gian trong chứng minh sự đúng đắn
  - Phân tích trạng thái của thuật toán
  - Phân tích sự ảnh hưởng của mỗi bước xử lý đến trạng thái của thuật toán

Chứng minh sự đứng đắn

- Ký hiệu
  - P - tiền điều kiện
  - Q - hậu điều kiện
  - A - thuật toán

  - A đúng đắn nếu với dữ liệu vào thoả mãn P thì A sẽ
    - cho kết quả thoả mãn Q
    - dừng sau một số bước xử lý hữu hạn
  - Ký hiệu

    {P} A {Q}

Chứng minh sự đứng đắn

- Các bước cơ bản

  - Xác định các tiền điều kiện và hậu điều kiện

  - Chú thích thuật toán bằng cánh chèn thêm các xác nhận liên quan đến trạng thái của thuật toán sao cho
    - tiền điều kiện được thoả mãn
    - xác nhận cuối cùng phải bao hàm hậu điều kiện

  - Chứng minh rằng mỗi bước xử lý, thuật toán đi từ xác nhận trước xử lý đến xác nhận sau xử lý

Các quy tắc chứng minh sự đúng đắn

- Một số quy tắc cho các cấu trúc lệnh cơ bản
  - Lệnh tuần tự
  - Lệnh điều kiện/rẽ nhánh
  - Lệnh lặp

Quy tắc lệnh tuần tự

Dãy lệnh tuần tự A	Quy tắc	Nghĩa là:
$\{P_0\}$ I_1 $\{P_1\}$ … $\{P_{i-1}\}$ I_k $\{P_i\}$ … $\{P_{n-1}\}$ I_n $\{P_n\}$	Nếu $P \Rightarrow P_0$ $\{P_{k-1}\}\ I_k\ \{P_k\}$, $k=2..n$ $P_n \Rightarrow Q$ Thì $\{P\}\ A\ \{Q\}$	Nếu -tiền điều kiện P bao hàm xác nhận đầu tiên -mỗi câu lệnh bao hàm xác nhận tiếp theo -xác nhận cuối cùng bao hàm hậu điều kiện Thì -dãy lệnh tuần tự A đúng

Quy tắc lệnh tuần tự

- Ví dụ
  - Hai biến x và y nhận hai giá trị tương ứng a và b. Hoán đổi giá trị hai biến x và y.
  - P = {x=a, y=b}
  - Q = {x=b, y=a}

```
{x=a, y=b, tmp chưa có giá trị}
tmp = x
{x=a, y=b, tmp=a}
x = y
{x=b, y=b, tmp=a}
y = tmp
{x=b, y=a, tmp=a}
```


Quy tắc lệnh điều kiện

Lệnh điều kiện A	Quy tắc	Nghĩa là:

$\{P_0\}$
if (c) then
    $\{c, P_0\}$
    I_1
    $\{c, P_1\}$
else
    $\{NOT\ c, P_0\}$
    I_2
    $\{NOT\ c, P_2\}$
endif
	Nếu
	    $P \Rightarrow P_0$
	    c có giá trị
	    c AND $P_1 \Rightarrow Q$
	    NOT c AND $P_2 \Rightarrow Q$
	Thì
	    {P} A {Q}
		Nếu
		-tiền điều kiện P bao hàm xác nhận đầu tiên
		-C có thể được định giá
		-cả hai nhánh đều bao hàm hậu điều kiện
		Thì
		-lệnh điều kiện A đúng

Quy tắc lệnh điều kiện

- Ví dụ
  - Tìm giá trị nhỏ nhất của a và b với a ≠ b
    preconditions: a ≠ b
    postconditions: m = min(a,b)

```
Lệnh A

{a ≠ b}
if (a < b) then
    {a < b}
    m = a
    {a < b, m = a}
else
    {a > b}
    m = b
    {a > b, m = b}
endif
```

{a < b, m = a} bao hàm m = min(a,b)

và

{a > b, m = b} bao hàm m = min(a,b)

Vậy {preconditions} A {postconditions}

Quy tắc lệnh lặp

- Một lệnh vòng lặp là đúng khi
  1. Nếu nó dừng, nó thoả mãn hậu điều kiện
  2. Nó dừng sau một số bước hữu hạn
- Nếu chỉ tính chất 1 đúng thì chỉ là *đúng đắn một phần*
- Đúng đắn một phần được chứng minh bởi quy nạp toán học hoặc *bất biến vòng lặp*
- Đúng đắn toàn phần cần chứng minh thêm thuật toán *dừng*

Bất biến vòng lặp

Lệnh lặp A

```
P ⇒ {I}
while (c) do
    {c, I}
    m = a
    {I}
endwhile
{NOT c, I} ⇒ Q
```

Định nghĩa

Một *bất biến vòng lặp* I là một *xác nhận* thoả mãn:

1. Bất biến vòng lặp đúng khi bắt đầu vòng lặp
2. Trong quá trình lặp (tức là điều kiện c đúng) thì bất biến vòng lặp I luôn đúng
3. Khi thoát khỏi vòng lặp (tức là điều kiện c sai) thì bất biến vòng lặp I phải bao hàm hậu điều kiện

Khi xác định được bất biến vòng lặp, nghĩa là đã chứng minh được thuật toán đúng đắn một phần

Bất biến vòng lặp

- Ví dụ
  - Tìm giá trị *m* nhỏ nhất trong mảng không rỗng x[1..n]

    preconditions: n ≥ 1

    postconditions: m = min(x[i] | 1 ≤ i ≤ n)

```
min (x[1..n])
begin
    m = x[1]
    for i from 2 to n do
        if (m > x[i]) then
            m = x[i]
        endif
    endfor
    return (m)
end
```

⇒

```
min (x[1..n])
begin
    i = 1, m = x[i]
    while (i < n) do
        i = i + 1
        if (m > x[i]) then
            m = x[i]
        endif
    endwhile
    return (m)
end
```

Bất biến vòng lặp

- Ví dụ
  P: $n \geq 1$
  Q: $m = \min(x[i] \mid 1 \leq i \leq n)$

```
min (x[1..n])
begin
   i = 1, m = x[i]
   {m = min(x[j], j=1..i)}
   while (i < n) do
      {i<n, m = min(x[j], j=1..i)}
      i = i + 1
      if (m > x[i]) then
         m = x[i]
      endif
      {m = min(x[j], j=1..i)}
   endwhile
   {i=n, m = min(x[j], j=1..i)}
   return (m)
end
```

Bất biến vòng lặp:
I = {m = min(x[j], j=1..i)}

Bởi vì:
- nếu i=1, m=x[1] thì I đúng
- nếu i<n, sau khi thực thi thân vòng lặp, I vẫn đúng
- nếu i=n, m=min(x[j], j=1..n) chính là hậu điều kiện

Hàm dừng

- Để chứng minh vòng lặp dừng sau một số bước lặp hữu hạn, chỉ cần xác định hàm dừng (termination function)
- Định nghĩa
  - Hàm T: N→N được gọi là hàm dừng nếu nó thoả mãn:
    1. T luôn giảm
    2. Nếu điều kiện *c* đúng thì T(p)>0, nếu T(p)=0 thì điều kiện *c* sai
- Nhận xét
  - T phụ thuộc biến đếm của vòng lặp p
    - Sau lần lặp thư nhất p = 1, sau lần lặp thứ hai p = 2, …
  - T sẽ bằng 0 vì nó luôn giảm
  - Khi T bằng 0 thì điều kiện c sai nên vòng lặp dừng

Hàm dừng

- Ví dụ

```
min (x[1..n])
begin
   i = 1, m = x[i]
   while (i < n) do
      i = i + 1
      {i_p = i_p-1 + 1}
      if (m > x[i]) then
         m = x[i]
      endif
   endwhile
   return (m)
end
```

Hàm dừng: $T(p) = n - i_p$

Bởi vì:

$T(p) = n - i_p = n - i_{p-1} - 1$
$= T(p-1) - 1$
Vậy $T(p) < T(p-1)$
Nghĩa là hàm T luôn giảm

Nếu điều kiện vòng lặp đúng,
thì $i_p < n$, vậy $T(p) > 0$
Nếu $T(p) = 0$, thì $n - i_p = 0$
Khi đó điều kiện vòng lặp sai

Ví dụ

- Tìm chỉ số của phần tử (1)
  - Cho mảng a[1..n] có chứa phần tử x. Tìm chỉ số i nhỏ nhất sao cho a[i] = x.
  - preconditions: $n \geq 1$, $\exists i \in [1..n]$: $a[i]=x$
  - postconditions: $\exists i \in [1..n]$: $a[i]=x$, $\forall k \in [1..i-1]: a[k] \neq x$
  - Chứng minh thuật toán sau là đúng

```
timphantu (a[1..n], x)
begin
   i = 1
   while (a[i] ≠ x) do
      i = i + 1
   endwhile
   return (i)
end
```

Ví dụ

- Tìm chỉ số của phần tử (2)
  - Xác định bất biến vòng lặp

```
timphantu (a[1..n], x)
begin
    i = 1
    {a[k] ≠ x, k=1..i-1}
    while (a[i] ≠ x) do
        {a[i] ≠ x, a[k] ≠ x, k=1..i-1}
        i = i + 1
        {a[k] ≠ x, k=1..i-1}
    endwhile
    {a[i] = x, a[k] ≠ x, k=1..i-1}
    return (i)
end
```

Bất biến vòng lặp:

$I = \{a[k] \neq x, k=1..i-1\}$

Chứng minh quy nạp:

-i=1, thì k=1..0 nên I đúng

-Giả sử I đúng sau bước lặp i, nghĩa là $a[k] \neq x$, k=1..i-1

-Ở bước lặp i+1:

-nếu $a[i+1] \neq x$ thì $a[k] \neq x$, k=1..i, nghĩa là I đúng

-nếu a[i+1]=x thì ta có hậu điều kiện Q

65

Ví dụ

- Tìm chỉ số của phần tử (3)
  - Xác định hàm dừng

Gọi k là chỉ số nhỏ nhất mà a[k]=x

```
timphantu (a[1..n], x)
begin
    i = 1
    while (a[i] ≠ x) do
        i = i + 1
        {i_p=i_(p-1)+1}
    endwhile
    return (i)
end
```

Hàm dừng:

$T = k - i_p$

Thật vậy:

$T(p) = k - i_p = k - i_{p-1} - 1$

$= T(p-1) - 1$

Vậy $T(p) < T(p-1)$

Nghĩa là hàm T luôn giảm

Nếu điều kiện $a[i_p] \neq x$ đúng, thì $k > i_p$, vậy $T(p) > 0$

Nếu $T(p) = 0$, thì $k = i_p$, khi đó điều kiện $a[i_p] \neq x$ sai

66

Nhận xét

- Dễ dàng đối với các lệnh tuần tự và lệnh điều kiện
- Khó khăn đối với lệnh lặp
- Xác định bất biến vòng lặp nói chung là rất phức tạp
- Chứng minh sự đúng đắn đòi hỏi nhiều thời gian và công sức
- Không thể áp dụng đối với các thuật toán phức tạp

67

Bài tập

1. Viết thuật toán tính n! và chứng minh nó đúng đắn.
2. Thuật toán sau làm gì ? Chứng minh câu trả lời.

```
begin
   m = n
   k = 0
   b[0] = m MOD 2
   m = m DIV 2
   while (m ≠ 0) do
      k = k + 1
      b[k] = m MOD 2
      m = m DIV 2
   endwhile
end
```

68

Bài tập

3. Thuật toán sau làm gì ? Chứng minh câu trả lời.

```
// cho b[1..n], ∀i: b[i] = 0 hoặc b[i] = 1
begin
    i = n
    m = b[i]
    while (i > 0) do
        i = i – 1
        m = 2*m + b[i]
    endwhile
end
```

4. Viết thuật toán tính giá trị trung bình các phần tử của mảng a[1..n]. Chứng minh thuật toán đúng đắn.

69

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
- 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

thời gian và bộ nhớ

- Đánh giá độ phức tạp
  - thời gian thực thi
  - bộ nhớ sử dụng
  - → thường mâu thuẩn 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

73

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

74

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

75

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
    - ...

76

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:
    $P(\underline{if}\ C\ \underline{then}\ I_1\ \underline{else}\ I_2) \leq P(C) + \max(P(I_1), P(I_2))$
  - 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

77

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

Vậy: C(n) = n

78

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 dộ phức xấu nhất

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 xuất xuất hiện của d

- Độ phức tạp trong trường hợp tốt nhất: $Cmin_A(n)$
  - $Cmin_A(n) = min\{C_A(d), d \in D_n\}$
- Độ phức tạp trong trường hợp xấu nhất: $Cmax_A(n)$
  - $Cmax_A(n) = max\{C_A(d), d \in D_n\}$
- Độ phức tạp trung bình: $Cmoy_A(n)$
  - $Cmoy_A(n) = \sum p(d).C_A(d)$

Tính độ phức tạp: ba trường hợp

- Nếu xác xuất xuất hiện các dữ liệu là như nhau, thì
  - $card(D_n) = |D_n|$
  - $Cmoy_A(n) = (1/|D_n|) \sum C_A(d)$
- Độ phức tạp trung bình nói chung là không thể xác định chính xác
- Điều chắc chắn
  - $Cmin_A(n) \leq Cmoy_A(n) \leq Cmax_A(n)$

81

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: $Cmin_A(n)$? $Cmoy_A(n)$? $Cmax_A(n)$

82

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 $L(n) = X$, $Cmax_A(n) = n$

83

Tính độ phức tạp: ba trường hợp

- Ví dụ 2: độ phức tạp trung bình
  - q: xác xuất 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 $C_A(D_{n,0}) = n$

$$Cmoy_A(n) = \sum_{i=0}^{n} p(D_{n,i})C_A(D_{n,i}) = (1-q)\,n + q/n \sum_{i=1}^{n} i$$

$$= (1-q)\,n + q(n+1)/2$$

84

Tính độ phức tạp: ba trường hợp

- Ví dụ 2: độ phức tạp trung bình
  - Nếu q = 1 (X thuộc L), thì $Cmoy_A(n) = (n+1)/2$
  - Nếu q = ½ (), thì $Cmoy_A(n) = (3n+1)/4$

  - 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

85

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

86

Các hàm tiệm cận

- Kí hiệu o nhỏ
  - f = o(g) khi n → ∞ 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):

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

  - Ví dụ
    - 2n = o(n^2)
    - n^2 = o(n^4)

Các hàm tiệm cận

- Kí hiệu O lớn
  - f = O(g) khi n → ∞ 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)

Các hàm tiệm cận

- Kí hiệu Θ
  - f = Θ(g) khi n → ∞ nếu:

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

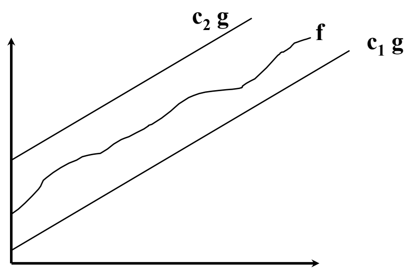
  - f(n) và g(n) tăng cùng tốc độ (so sánh trong nghĩa độ phức tạp)

  - 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: $c_1 <= 1/2 - 3/n <= c_2$
    - Với $c_2 = 1/2$, $n_0 = 12$, c1 = 1/2 – 3/12 = 1/4
    - Vậy $n^2/2 - 3n = \Theta(n_2)$

89

Các hàm tiệm cận

- Kí hiệu Θ
  - định nghĩa tiệm cận trên và tiện cận dưới của f(n)



  - Nếu f = Θ(g) thì:
    - f = O(g)
    - g = O(f)

90

Các hàm tiệm cận

- Kí hiệu Ω
  - f = Ω(g) khi n $\to \infty$ nếu:

$$\exists c>0, \exists n_0>0, 0 \le cg(n) \le f(n), \forall n \ge n_0$$

  - 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
    - $n^2 = \Omega(n)$

Các hàm tiệm cận

- Các tính chất
  - Nếu f = Θ(g) thì f = O(g) và f = Ω(g)
    - Suy ra từ định nghĩa
  - Nếu f = Θ(g) thì g = Θ(f)
    - Chứng minh

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

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

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

Các hàm tiệm cận

- Hàm tương đương
  - f ~ g khi n → ∞ nếu:

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

  - 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) ~ g(n)

93

Các hàm tiệm cận

- Độ phức tạp lý thuyết khi n → ∞

f = o(g)	độ phức tạp của f < độ phức tạp của g
f = O(g)	độ phức tạp của f ≤ độ phức tạp của g (theo các hằng số xác định)
f = Θ(g)	độ phức tạp của f cùng cở độ phức tạp của g (theo các hằng số xác định)
f = Ω(g)	độ phức tạp của f ≥ độ phức tạp của g (theo các hằng số xác định)
f = ~(g)	độ phức tạp của f = độ phức tạp của g

94

Các hàm tiệm cận

- Ví dụ 1
  - Giả sử $Cmin_A(n) = 8n^2 + 2n + 1$ và $Cmin_B(n) = 1000n^2 + 10n + 2$ là các hàm biểu diễn thoài 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.
  - $Cmin_A(n) = 8n^2 + 2n + 1$
    $\le 8n^2 + 2n^2 + n^2$ với $n \ge 1$
    $= 11n^2$
    chọn $c = 11$, $n_0 = 1$, thì $Cmin_A(n) = O(n^2)$
  - $Cmin_B(n) = 1000n^2 + 10n + 2$
    $\le 1000n^2 + 10n^2 + 2n^2$ với $n \ge 1$
    $= 1012n^2$
    chọn $c = 1012$, $n_0 = 1$, thì $Cmin_B(n) = O(n^2)$
  - Thuật toán: $O(n^2)$

Các hàm tiệm cận

- Ví dụ 2
  - Chứng minh rằng $\forall k \in N$: $\sum_{i=1}^{n} i^k = \theta(n^{k+1})$
  - Ta có: $\sum_{i=1}^{n} i^k \le \sum_{i=1}^{n} n^k = n.n^k = n^{k+1}$
  - Vậy: $\sum_{i=1}^{n} i^k = O(n^{k+1})$ với $c = 1$
  - Ta có: $\sum_{i=1}^{n} i^k \ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} i^k \ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} (\frac{n}{2})^k \ge \frac{n}{2}(\frac{n}{2})^k = \frac{n^{k+1}}{2^{k+1}}$
  - Vậy: $\sum_{i=1}^{n} i^k = \Omega(n^{k+1})$ với $c = 1/2^{k+1}$

Các hàm tiệm cận

- Bài tập
  - Giả sử hàm f(n) = log (2n + α) được định nghĩa, chứng minh rằng: f(n) = Θ(log(n))

97

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: $a_n = b_1 a_{n-1} + b_2 a_{n-2} + \ldots + b_k a_{n-k} + g(n)$
    - tuyến tính thuần nhất hệ số hằng số:

      $a_n = b_1 a_{n-1} + b_2 a_{n-2} + \ldots + b_k a_{n-k}$
    - không tuyến tính: ...

      k hạng đầu tiên phải có giá trị

98

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

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

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

$a_{n-1} = b\, a_{n-2} + g_{n-1}$ * với hệ số = b

$a_{n-2} = b\, a_{n-3} + g_{n-2}$ * với hệ số = b^2

…

$a_1 = b\, a_0 + g_1$ * với hệ số = b^{n-1}

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

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

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

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

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

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

Hệ thức truy hồi

- Truy hồi tuyến tính bậc nhất (3)
  - Ví dụ tính:
    - $a_n = 4 a_{n-1} - 1$, $a_0 = 1/3$
    - $a_n = 4 a_{n-1} - 1$, $a_0 = 0.333$
    - $a_n = 3 a_{n-1}^2$ et $a_0 = 1$
      có thể đặt $b_n = \log_2 a_n$

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

    $r^2 - br - c = 0$ (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 của (1), với α và β là các hằng số
    - Chứng minh …

103

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: $a_n = 3.2^n -(-1)^n$

104

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: $a_n = 2^n + 3.2^n$

105

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

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

106

Hệ thức truy hồi

- Truy hồi phân hoạch (2)
  - Thường được xét dưới dạng:

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

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

$t_0 = 1\ (= a_1 = 1)$

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

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

….

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

Hệ thức truy hồi

- Truy hồi phân hoạch (3)

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

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

$t_{k-2} = c\ t_{k-3} + g(b^{k-2})$ *c²

…

$t_1 = c\ t_0 + g(b^1)$ *c^{k-1} $t_0=1$

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

với $n = b^k \Leftrightarrow k = \log_b n$

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

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

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

Hệ thức truy hồi

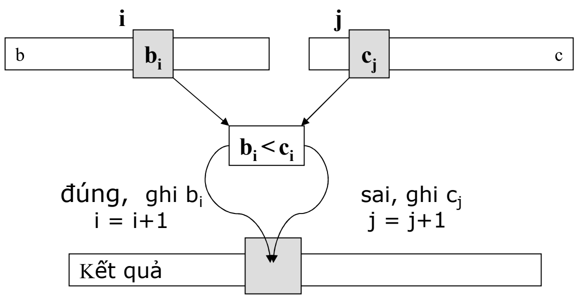
- Ví dụ (1)
  - Tính độ phức tạp của thuật toán sắp xếp trộn (merge sort)

```
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 $\lceil n/2 \rceil$

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

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

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

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

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

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

$$\boxed{a_n = n\log_2 n + 1}$$

Vậy: C(n) = Θ(nlog(n))

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)

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,…n, kí hiệu t_j là số lần thực thi lệnh lặp While

insertion-sort(A)	cost	times
Begin		
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		
End		

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:

$$C(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= n(c_1 + c_2 + c_3 + c_4 + c_7) - (c_2 + c_3 + c_4 + c_7)$$

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

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

115

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 phân 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 nữa số phần tử của A[1..j-1] lớn hơn A[j] và một nữa 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

116

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: $\Theta(n^2)$

117

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 $\leq O(n^3)$

118

Các lớp thuật toán theo độ phức tạp

- Minh hoạ 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}	∞

Bài tập

- Bài 1
  - 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

Bài tập

- Bài 2
  - 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
```

121

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

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

    - 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
```

123

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 Fibonaccci

$F_0 = 1, F_1 = 1$

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

    - 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
```

124

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/lẽ

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

    - 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

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

    - 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 liền 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

127

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 tuc giảm cho đến khi a≥b thì sẽ dừng, trường hợp đã được xác định

128

Đệ quy

- Ví dụ 2
  - Thuật toán

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

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

129

Đệ quy

- Không thể xác định tính dừng (1)
  - Vấn đề
    - Có thể xây dựng chương trình tự động kiểm tra một chương trình P có thể dừng khi thực thi trên bộ dữ liệu D ?
    - Vào
      - chương trình P
      - bộ dữ liệu D
    - Ra
      - đúng, nếu chương trình P dừng trên bộ dữ liệu D
      - Sai, nếu ngược lại

130

Đệ quy

- Không thể xác định tính dừng (2)
  - Giả sử tồn tại chương trình *terminate* kiểm tra tự động tính dừng của một chương trình
  - 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 !

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

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

133

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

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

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

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

135

Đệ quy

- Chi phí đệ quy (1)
  - Chi phí lời gọi hàm

…
f(x, y, z)
…

f(a, b, c)
…
end

1. Lưu lại ngữ cảnh
2. Tạo ngăn xếp lưu các tham số, địa chỉ quay về, …
3. Chuyển đến hàm
4. Lấy từ ngăn xếp
5. Thực thi
6. Tạo ngăn xếp chứa kết quả trả về
7. Quay về vị trí gọi hàm
8. Lấy kết quả từ ngăn xếp
9. Phục hồi lại ngữ cảnh

136

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

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 (terminal 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 terminal recursion)
    - Thuật toán được gọi là đệ quy không kết thúc nếu nó chứa các xử lý nào sau lời gọi đệ quy

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(α(U))
   else
            T
   endif
end
```

```
P'(U)
begin
   while C do
            D
            U = α(U)
   endwhile
   T
end
```

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

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
```

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
            endif
  endwhile
  bsearch' = 0
end
```


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

Khử đệ quy

- Đệ quy không kết thúc (2)
  - Cách 1

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

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


Khử đệ quy

- Đệ quy không kết thúc (3)
  - Cách 1
    - Minh hoạ

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
         $T(\alpha(\alpha(U_0)))$
      $E(\alpha(U_0))$
   $E(\alpha(\alpha(U_0))$

Gọi $Q'(U_0)$?

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)
            print(n) //in n
  endif
end
```

```
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 hoạ

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
      $T(\alpha(\alpha(U_0)))$
    $E(\alpha(U_0))$
  $E(\alpha(\alpha(U_0))$

Gọi $Q'(U_0)$?

147

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

149

Khử đệ quy

- Bài tập (1)
  - Bài 1
    - Định nghĩa dãy số Fibonaccci

      $Fib_0 = 1, Fib_1 = 1$

      $Fib_n = Fib_{n-1} + Fib_{n-2}$
    - Hãy thực hiện
      1. Xây dựng thuật toán đệ quy tính 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 (Fib(n), Fib(n-1)) với n > 0
      4. Sử dụng thuật toán trong câu 3 để xây dựng thuật toán mới tính Fib(n)
      5. Đánh giá độ phức tạp (bởi số phép cộng) của thuật toán trên

150

Khử đệ quy

- 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

Chia để 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 đặt ra

Chia để trị

- Cấu trúc chung

```
chia-để-trị(x: bài toán) : giải pháp
begin
   if (x nhỏ và đơn giản) then
           return (thuật toán đơn giản)
   else
           phân tích x thành nhiều bài toán con x1, x2, … xk
           for i from 1 to k do
               yi = chia-để-trị(xi)
           endfor
           kết hợp các giải pháp yi thành giải pháp y của x
           return (y)
  endif
end
```

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 lồi

155

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 (S[i] > max) then max = S[i]
         endif
         if (S[i] < min) then min = S[i]
         endif
   endfor
   return (max, min)
end
```

156

Tìm giá trị lớn nhất và giá trị nhỏ nhất

- Thuật toán chia để trị

```
maxmin(A,x,y)
begin
  if (y-x ≤ 1)
     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
```

157

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 luỹ thừa của 2, nghĩa là y – x lẽ và x+y cũng lẽ
    - 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

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

158

Tìm giá trị lớn nhất và giá trị nhỏ nhất

- Phân tích thuật toán
  - Tính số phép so sánh

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

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

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 k 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

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

161

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)
  endif
end
```

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

162

Nhân hai ma trận

- Thuật toán chia để trị (3)

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

$\Rightarrow$ $\boxed{O(n^3)}$

Nhân hai ma trận

- Thuật toán Strassen (1)

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)

- Khi đó

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

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

Nhân hai ma trận

- Thuật toán Strassen (2)
  - Tại sao đúng

$r = m_2+m_3-m_6-m_7$
$= (b+d)(g+h) + (a-d)(e+h) - (a+b)h - d(g-e)$
$= bg+bh+dg+dh+ae+ah-de-dh-ah-bh-dg+de$
$= ae + bg$

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

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

$u = m_1-m_3-m_4-m_5$
$= (a+c)(e+f) - (a-d)(e+h) - a(f-h) - (c+d)e$
$= ae+af+ce+cf-ae-ah+de+dh-af+ah-ce-de$
$= cf + dh$

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
```

Nhân hai ma trận

- Thuật toán Strassen (4)
  - Độ phức tạp

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

    $C(1) = 1$

Nhân hai ma trận

- Thuật toán Strassen (5)
  - Độ phức tạp

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

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 A1, A2, A3 sao cho các phần tử
        của A1 nhỏ hơn x, các phần tử của A2 bằng x và các phần
        tử của A3 lớn hơn x
        return (quicksort(A1), A2, quicksort(A3))
  endif
end
```

169

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
    - $|A_1| = i-1$
    - $|A_2| = 1$
    - $|A_3| = n-1$
  - Giả sử xác xuất phần tử x nhỏ thứ i trong A là 1/n

170

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 ∈ [1..n] với xác xuất 1/n
  - Để chia A thành A_1, A_2 và A_3 cần n-1 phép so sánh
  - Vậy, với n ≥ 2

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

  - Mà

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

  - Vậy, với n ≥ 2

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

Quicksort

- Độ phức tạp trong trường hợp trung bình (3)
  - Nhân hai vế của (1) với n, với n ≥ 2 có

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

  - Thay n bởi n-1, với n ≥ 3 có

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

  - Lấy (2) trừ (3)

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

  - Vậy

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

  - Chia hai vế của (4) cho n(n+1)

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

Quicksort

- Độ phức tạp trong trường hợp trung bình (4)
  - Đặt S(n) = C(n)/(n+1), từ (5) ta có, với n≥3

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

với S(0) = S(1) = 0

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

  - Vậy

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

$$\begin{aligned} S(n) &\le S(n-1) + 2/n \le S(n-2) + 2/(n-1) + 2/n \\ &\le S(n-3) + 2/(n-2) + 2/(n-1) + 2/n \\ &\le S(n-i) + 2\sum_{k=n-i+1}^{n} 1/k \end{aligned}$$

173

Quicksort

- Độ phức tạp trong trường hợp trung bình (5)
  - Thay i = n-1, ta có

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

  - Vậy

$$\begin{aligned} C(n) &= (n+1)S(n) \\ &\le 2(n+1)\ln n \\ &\le 2(n+1)\frac{\log n}{\log e} \\ &\le 1.386(n+1)\log n \end{aligned}$$

  - Vậy O(nlog(n))

174

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

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

  - Vậy: $\Theta(n^2)$

175

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 ~ n/2, nghĩa là i=n/2

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

  - Vậy: nlog(n)+O(n)

176

Quicksort

- Cài đặt

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

177

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(nlogn)*, độ phức tạp bài toán chọn phần tử cũng là *O(nlogn)*
- Tồn tại giải pháp tốt hơn ?

178

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

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)
  S1 = {các phần tử trung bình từ các danh sách con}, |S1|=m
  x0 = F(S1, m/2) // phần tử trung bình của các phần tử trung bình
  S2 = {x ∊ S| x < x0}, S3 = {x ∊ S| x > x0}
  if (|S2| ≤ k) then return (F(S2, k))
  else if (|S3 ≥ n-k+1|) then return (F(S3, n-k+1))
       else return (x0)
       endif
  endif
end
```

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

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

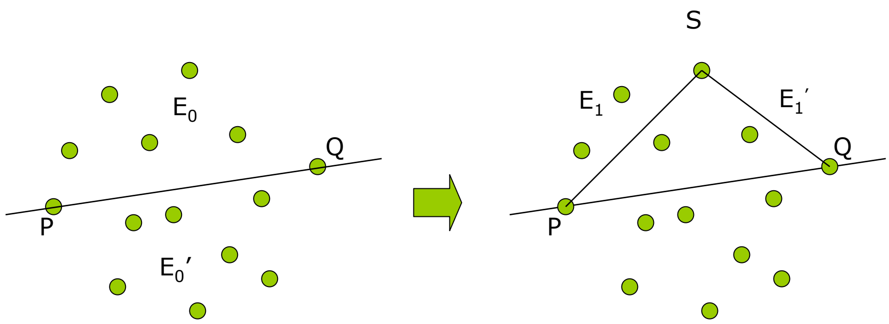
Tính bao đóng lồi

- 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'

183

Tính bao đóng lồi

- Minh hoạ



184

Tính bao đóng lồi

- Thuật toán (1)

```
bao-dong-loi (E)
begin
    Tính P và Q
    if (hoành độ P = hoành độ Q) then
        // tất cả n điểm trên đường thẳng
        return (danh sách các điểm của E)
    else
        Tính E0 và E0'
        return (hoa-nhap(nua-baoloi(E0, P, Q), nua-baoloi(E0', Q, P)))
    endif
end
```


Tính bao đóng lồi

- 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 E1 = là phần giới hạn bởi PS không chứa Q
        Tính E1'= là phần giới hạn bởi QS không chứa P
        return (hoa-nhap(nua-baoloi(E1, P, S), nua-baoloi(E1', S, Q)))
    endif
end
```

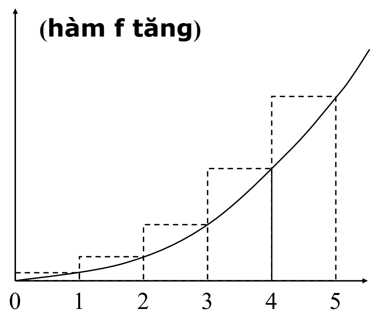
Tính bao đóng lồi

- 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)

Tính tổng

- Tính tổng (1)

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



Ví dụ: f(x)=i^p

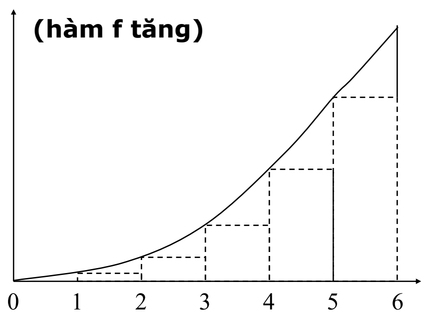
$$\int_0^5 f(t)dt \le \sum_{i=1}^{5} f(i)$$

Tổng quát

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

Tính tổng

- Tính tổng (2)



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

Tổng quát

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

Vậy: $$\int_{0}^{n} f(t)dt \le \sum_{i=1}^{n} f(i) \le \int_{1}^{n+1} f(t)dt$$

Nếu hàm f giảm: $$\int_{1}^{n+1} f(t)dt \le \sum_{i=1}^{n} f(i) \le \int_{0}^{n} f(t)dt$$

189

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 nữa có bậc n/2 (mũ [0..n/2]) và một nữa có bậc n (mũ [n/2+1..n])
     b. Thuật toán thứ hai cần phải chia đôi đa thức thành hai đa thức, một nữa có mũ là chẵn, một nữa có mũ là lẽ

  3. Chứng minh rằng hai số nguyên được biểu diễn bởi n bít có thể được nhân bởi thuật toán có độ phức tạp $O(n^{\log_2^3})$

190

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

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

Quy hoạch động

- Quy hoạch động = Chia để trị + mảng

- Chia để trị : tiếp cận từ trên xuống
  - Giải quyết bài toán lớn trước sau đó giải quyết bái 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 bái toán lớn sau

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

195

Một số ứng dụng

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

196

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

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

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

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

197

Triển khai nhị thức $(a+b)^n$

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

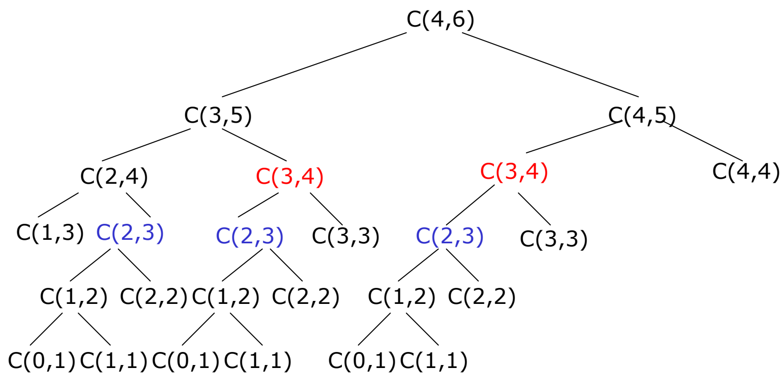
```
C(k, n)
begin
    if (k = 0 or k = n) then
          return (1)
    else
          return (C(k-1, n-1) + C(k, n-1))
end
```

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

198

Triển khai nhị thức $(a+b)^n$

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



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

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

với c và d là các hằng số

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: $C(n) = \Theta(2^n)$

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

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

    - 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

Triển khai nhị thức $(a+b)^n$

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

```
C1(k, n)
begin
    // sử dụng mảng hai chiều C[0..n,0..k]
    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
```

203

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**

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

204

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)
      $k(n-k) = nk - k^2 \le nk$
    - 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 ?

205

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

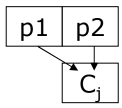
```
C2(k, n)
begin
   // sử dụng mảng một chiều C[0..n]
   C[0] = 1 // khởi gán hàng 1
   C[1] = 1
   // tính từng hàng
   for i from 2 to n do
        p1 = 1
        for j from 1 to i-1 do
            p2 = C[j]
            C[j] = p1 + p2
            p1 = p2
        endfor
        C[i] = 1 // phần tử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

206

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



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)

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

    - 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)

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 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

209

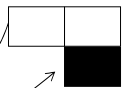
Thiết kế thuật toán quy hoạch động

Chia để trị

```
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
```

Quy hoạch động

```
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
```



210

Nhân dãy ma trận

- Bài toán
  - Có n ma trận M_1, M_2, … M_n, cần tính tích $M_1.M_2.....M_n$ sao cho thực hiện ít phép nhân nhất
  - Phép nhân ma trận có tính kết hợp
    - Có thể thực hiện tính tích $M_1.M_2.....M_n$ bởi nhiều thứ tự kết hợp khác nhau
  - Giả sử dụng thuật toán đơn giản để nhân hai ma trận
    - Nhân hai ma trận có kích thước *pxq* và *qxr* cần thực hiện *pxqxr* phép nhân

211

Nhân dãy ma trận

- Ví dụ
  - Nhân dãy ma trận
    $M_1(10x20).M_2(20x50).M_3(50x1).M_4(1x100)$
  - Có các 5 cách kết hợp
    - $(M_1.(M_2.(M_3.M_4)))$: 50x1x100+20x50x100+10x20x100=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?

212

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 ≥ 2 thì có thể chia tích $M_1.M_2.....M_n$ thành hai

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

có n-1 cách chia (k=1..n-1)

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

$$P(n) = \Omega(2^n)$$

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 $M_iM_{i+1}...M_j$
    - Rỏ ràng $m_{i,i} = 0$
  - Giả sử kích thước các ma trận $M_i,M_{i+1},...,M_j$ lần lượt là (d_{i-1},d_i), (d_i,d_{i+1}), ..., (d_{j-1},d_j)
  - Giả sử rằng, chúng ta biết được cách kết hợp $M_iM_{i+1}...M_j$ với chi phí tối thiểu ($m_{i,j}$ nhỏ nhất) là $(M_iM_{i+1}...M_k).(M_{k+1}...M_j)$
    - nghĩa là k đã biết
    - khi đó:

      $m_{i,j} = m_{i,k} + m_{k+1,j}$ + chi phí nhân hai ma trận $(d_{i-1},d_k).(d_k,d_j)$

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

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

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)
// sử dụng mảng m[i,j] lưu trữ giá trị mi,j
// ban đầu, m[i,j] được khởi gán +∞
begin
    if (i=j) then return (0)
    else for k from i to j-1 do
            r = matrixChainRecur(i,k) + matrixChainRecur(k+1,j) + di-1dkdj
            if (r < m[i, j]) then
                    r = m[i,j]
            endif
        endfor
   return (m[i,j])
end
```

Nghiệm của bai toán là m[1,n]

Nhân dãy ma trận

- Thuật toán chia để trị (3)
  - Đánh giá độ phức tạp
    - Gọi C(n) là thời gian thực hiện chainMatrixRecur(i,j) khi i=1, j=n
    - Ta có

$$C(n) = \begin{cases} c & n = 0 \\ \sum_{k=1}^{n-1} \left(C(k) + C(n-k) + d \right) & n > 0 \end{cases}$$

với c, d là các hằng số

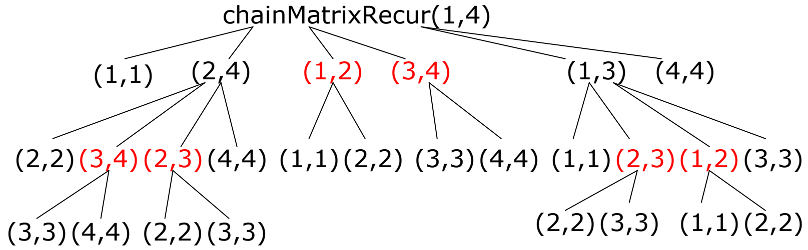
- Hay

$$C(n) = 2\sum_{k=1}^{n-1} C(k) + d(n-1)$$

- Vậy C(n) ≥ 2C(n-1), nghĩa là C(n) = $\Omega(2^n)$
- Nếu tính chính xác, thì C(n) = $\Theta(3^n)$

Nhân dãy ma trận

- Thuật toán chia để trị (4)
  - Ví dụ



Có nhiều bài toán con được thực hiện nhiều lần !

Nhân dãy ma trận

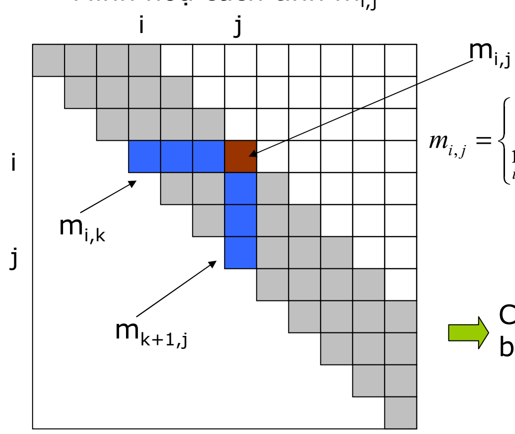
- Thuật toán quy hoạch động (1)
  - Chúng ta cần xây dựng mảng m[1..n,1..n]
  - Với mỗi m[i,j], i≤j, nên chỉ nữa trên đường chéo chính của mảng m được sử dụng
  - Mỗi m[i,j] được tính theo công thức

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

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

Nhân dãy ma trận

- Thuật toán quy hoạch động (2)
  - Minh hoạ cách tính $m_{i,j}$



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

⇨ Cần phải xây dựng dần bảng m theo các đường chéo

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

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

  - 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

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 m_i,j = min(m_i,k+m_k+1,i+s+d_i-1 d_k d_i+s)
           m[i,j] = +∞
           for k from i to j-1 do
                r = m[i,k] + m[k+1,j] + d_i-1 d_k d_j
                if (r < m[i,j]) then
                    r = m[i,j]
                    g[i,j] = k  // ghi nhớ k
                endif
           endfor
       endfor
    endfor
    return m và s
end
```

Nhân dãy ma trận

- Thuật toán quy hoạch động (5)
  - Ví dụ
    - $M_1(10x20).M_2(20x50).M_3(50x1).M_4(1x100)$
      nghĩa là: $d_0=10$, $d_1=20$, $d_2=50$, $d_3=1$, $d_4=100$
    - Xây dựng đường chéo s = 1
      - $m[1,2] = \min(m[1,k] + m[k+1,2] + d_0d_1d_2)$, với $1 \le k < 2$
        $= \min(m[1,1] + m[2,2] + d_0d_1d_2)$
        $= d_0d_1d_2 = 10x20x50 = 10000$
        g[1,2] = k = 1
      - $m[2,3] = \min(m[2,k] + m[k+1,3] + d_1d_2d_3)$, với $2 \le k < 3$
        $= d_1d_2d_3 = 20x50x1 = 1000$
        g[2,3] = k = 2
      - $m[3,4] = d_2d_3d_4 = 50x1x100 = 5000$
        g[3,4] = k = 3

Nhân dãy ma trận

- Thuật toán quy hoạch động (6)
  - Ví dụ
    - Xây dựng đường chéo s = 2
      - $m[1,3] = \min(m[1,k] + m[k+1,3] + d_0d_kd_3)$, với $1 \le k < 3$
        $= \min(m[1,1] + m[2,3] + d_0d_1d_3,$
        $m[1,2] + m[3,3] + d_0d_2d_3)$
        $=\min(0 + 1000 + 200, 10000 + 0 + 500)$
        $= 1200$
        g[1,3] = k = 1
      - $m[2,4] = \min(m[2,k] + m[k+1,4] + d_1d_kd_4)$, với $2 \le k < 4$
        $= \min(m[2,2] + m[3,4] + d_1d_2d_4,$
        $m[2,3] + m[4,4] + d_1d_3d_4)$
        $= \min(0 + 5000 + 100000, 1000 + 0 + 2000)$
        $= 3000$
        g[2,4] = k = 3

Nhân dãy ma trận

- Thuật toán quy hoạch động (7)
  - Ví dụ
    - Xây dựng đường chéo s = 3
      - m[1,4] = min(m[1,k] + m[k+1,4] + $d_0d_kd_4$), với $1 \le k < 4$
        = min(m[1,1] + m[2,4] + $d_0d_1d_4$,
        m[1,2] + m[3,4] + $d_0d_2d_4$,
        m[1,3] + m[4,4] + $d_0d_3d_4$)
        = min(0 + 3000 + 20000,
        10000 + 5000 + 50000,
        1200 + 0 + 1000)
        = 2200
        g[1,4] = k = 3
    - m[1,4] chính là kết quả cần tìm

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

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 hoach động tốt hơn so với thuật toán đơn giản và thuật toán chia để trị

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 $M_iM_{i-1}...M_j$ 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)

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, s[i,j])
       Y = chainMatrixProduct(M, g, s[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

229

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: $M_1(10x20).M_2(20x50).M_3(50x1).M_4(1x100)$

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

230

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 X = 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

Dãy con chung dài nhất

- Thuật toán vét cạn
  - Giả sử $X=x_1x_2\ldots x_n$ và $Y = y_1y_2\ldots y_m$
  - So sánh mỗi dãy kí hiệu con của 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 !

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 $X=x_1x_2 \dots x_n$, $X_i = X=x_1x_2 \dots x_i$ được gọi là tiền tố thứ *i* của X
  - Gọi $Z = z_1z_2 \dots z_k$ là dãy con chung dài nhất (LCS) của $X=x_1x_2 \dots x_n$ và $Y = y_1y_2 \dots y_m$

    Nếu $x_n=y_m$ thì $z_k=x_n=y_m$ và $Z_{k-1} = LCS(X_{n-1},Y_{m-1})$

    Nếu $x_n \neq y_m$ và $z_k \neq x_n$ thì $Z = LCS(X_{n-1},Y)$

    Nếu $x_n \neq y_m$ và $z_k \neq y_m$ thì $Z = LCS(X,Y_{m-1})$

233

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

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

234

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 (x_i = y_j) 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
```

235

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 $\Omega(2^m)$

236

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 (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])
        endif
     endfor
  endfor
  return c
end
```


Dãy con chung dài nhất

- Thuật toán quy hoạch động (2)
  - Ví dụ

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

X=ABCB, Y=BDCAB

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**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

<u>for</u> i <u>from</u> 0 <u>to</u> n <u>do</u> c[i,0]=0
<u>for</u> j <u>from</u> 0 <u>to</u> m <u>do</u> c[0,j]=0

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**	**0**
1	**A**	**0**	**0**				
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

<u>if</u> (xi = yj) <u>then</u> c[i,j] = c[i-1,j-1] + 1
<u>else</u> c[i,j] = max(c[i-1,j], c[i,j-1])

Dãy con chung dài nhất

- Thuật toán quy hoạch động (5)
  - Ví dụ

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

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])

241

Dãy con chung dài nhất

- Thuật toán quy hoạch động (6)
  - Ví dụ

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

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])

242

Dãy con chung dài nhất

- Thuật toán quy hoạch động (7)
  - Ví dụ

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

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])

243

Dãy con chung dài nhất

- Thuật toán quy hoạch động (7)
  - Ví dụ

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

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])

244

Dãy con chung dài nhất

- Thuật toán quy hoạch động (8)
  - Ví dụ

i	j	0	1	2	3	4	5
	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**	
3	**C**	**0**					
4	**B**	**0**					

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])

245

Dãy con chung dài nhất

- Thuật toán quy hoạch động (9)
  - Ví dụ

i	j	0	1	2	3	4	5
	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**					
4	**B**	**0**					

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])

246

Dãy con chung dài nhất

- Thuật toán quy hoạch động (10)
  - Ví dụ

i	j	0	1	2	3	4	5
		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			
4	B	0					

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])

247

Dãy con chung dài nhất

- Thuật toán quy hoạch động (11)
  - Ví dụ

i	j	0	1	2	3	4	5
		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 (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])

248

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 (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])

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 (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])

Dãy con chung dài nhất

- Thuật toán quy hoạch động (14)
  - Ví dụ

i		j: 0	1	2	3	4	5
		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 (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])

Dãy con chung dài nhất

- Thuật toán quy hoạch động (15)
  - Ví dụ

i		j: 0	1	2	3	4	5
		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 (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])

Dãy con chung dài nhất

- Thuật toán quy hoạch động (16)
  - Đánh giá độ phức tạp
    - Thuật toán nhằm tính mảng c có *nxm* phần tử
    - Tính mỗi phần tử cần O(1) thời gian
    - Vậy độ phức tạp của thuật toán là O(nm)

253

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

254

Dãy con chung dài nhất

- Thuật toán quy hoạch động (18)
  - Ví dụ

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	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

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

255

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
```

256

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
```

257

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

W = 12

Ba lô

Đồ vật	Trọng lượng - w_i	Giá trị - b_i
	2	8
	5	10
	3	6
	9	2
	4	3

258

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 !

259

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

260

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 $w-w_k$

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

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

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

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)
       return (max(x, y+bk))
     endif
  endif
end
```

Sử dụng: balo(n, W)

263

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 phương trình truy hồi

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

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

với *c* và *d* là các hằng số

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

264

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,w] = 0 với mọi k
  - Sau đó, v[k,w] sẽ được tính theo v[k-1,w] hoặc v[k-1,w-w_k]

265

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 (wk ≤ w) then // có thể sử dụng đò vật k
              v[k,w] = max(v[k-1,w], v[k-1,w-wk] + bk)
           else // wk > w
              v[k,w] = v[k-1,w] // không sử dụng đồ vật k
          endif
       endfor
   endfor
end
```

266

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 (w_k ≤ w) then // có thể sử dụng đồ vật k
              if (b_i + v[i-1,w-w_i] > v[i-1,w]) then
                 v[i,w] = b_i + v[i-1,w- w_i] // sử dụng đồ vật k
              else
                 v[i,w] = v[i-1,w] // không sử dụng đồ vật k
              endif
           else // w_k > w
              v[k,w] = v[k-1,w] // không sử dụng đồ vật k
          endif
       endfor
   endfor
end
```


Xếp ba lô

- Thuật toán quy hoạch động (4)
  - Đánh giá độ phức tạp
    - Thuật toán tính mảng *nxW* phần tử bởi hai vòng lặp lồng nhau
    - Độ phức tạp O(nW)
  - Ví dụ minh hoạ

    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)

Xếp ba lô

Ví dụ (2)

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

<u>for</u> w = 0 <u>to</u> W <u>do</u> v[0,w] = 0

Xếp ba lô

Ví dụ (3)

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

<u>for</u> k = 0 <u>to</u> n <u>do</u> v[k,0] = 0

Xếp ba lô

Ví dụ (4)

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

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

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

```
if wk <= w then // đồ vật k có thể được sử dụng
    if bk + v[k-1,w-wk] > v[k-1,w] then
        v[k,w] = bk + v[k-1,w- wk]
    else
        v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

271

Xếp ba lô

Ví dụ (5)

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

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

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

```
if wk <= w then // đồ vật k có thể được sử dụng
    if bk + v[k-1,w-wk] > v[k-1,w] then
        v[k,w] = bk + v[k-1,w- wk]
    else
        v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

272

Xếp ba lô

Ví dụ (6)

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

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

Đồ vật

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

```
if wk <= w then // đồ vật k có thể được sử dụng
   if bk + v[k-1,w-wk] > v[k-1,w] then
     v[k,w] = bk + v[k-1,w- wk]
   else
     v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

273

Xếp ba lô

Ví dụ (7)

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

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

Đồ vật

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

```
if wk <= w then // đồ vật k có thể được sử dụng
   if bk + v[k-1,w-wk] > v[k-1,w] then
     v[k,w] = bk + v[k-1,w- wk]
   else
     v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

274

Xếp ba lô

Ví dụ (8)

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

k=1

b_k=3

w_k=2

w=5

w-w_k=2

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if w_k <= 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

275

Xếp ba lô

Ví dụ (9)

W \ k	0	1	2	3	4
0	0	0	0	0	0
1	0	0 →	**0**		
2	0	3			
3	0	3			
4	0	3			
5	0	3			

k=2

b_k=4

w_k=3

w=1

w-w_k=-2

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if w_k <= 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

276

Xếp ba lô
Ví dụ (10)

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

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

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

```
if wk <= w then // đồ vật k có thể được sử dụng
   if bk + v[k-1,w-wk] > v[k-1,w] then
      v[k,w] = bk + v[k-1,w- wk]
   else
      v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

277

Xếp ba lô
Ví dụ (11)

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

$k=2$
$b_k=4$
$w_k=3$
$w=3$
$w-w_k=0$

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

```
if wk <= w then // đồ vật k có thể được sử dụng
   if bk + v[k-1,w-wk] > v[k-1,w] then
      v[k,w] = bk + v[k-1,w- wk]
   else
      v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

278

Xếp ba lô
Ví dụ (12)

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

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

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

if w_k <= 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

279

Xếp ba lô
Ví dụ (13)

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

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

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

if w_k <= 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

280

Xếp ba lô

Ví dụ (14)

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

$k=3$
$b_k=5$
$w_k=4$
$w=1..3$
$w- w_k<0$

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

```
if wk <= w then // đồ vật k có thể được sử dụng
    if bk + v[k-1,w-wk] > v[k-1,w] then
        v[k,w] = bk + v[k-1,w- wk]
    else
        v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

281

Xếp ba lô

Ví dụ (15)

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

$k=3$
$b_k=5$
$w_k=4$
$w=4$
$w- w_k=0$

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

```
if wk <= w then // đồ vật k có thể được sử dụng
    if bk + v[k-1,w-wk] > v[k-1,w] then
        v[k,w] = bk + v[k-1,w- wk]
    else
        v[k,w] = v[k-1,w]
else v[k,w] = v[k-1,w] // wk > w, đồ vật k không được sử dụng
```

282

Xếp ba lô

Ví dụ (16)

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

k=3
b_k=5
w_k=4
w=5
w-w_k=1

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

```
if w_k <= 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
```


Xếp ba lô

Ví dụ (17)

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	

k=4
b_k=6
w_k=5
w=1..4
w-w_k<0

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

```
if w_k <= 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
```

Xếp ba lô

Ví dụ (18)

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

$k=4$
$b_k=6$
$w_k=5$
$w=5$
$w-w_k=0$

Đồ vật

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

```
if w_k <= 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
```


Xếp ba lô

- Thuật toán *balo* 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

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

Các đồ vật sử dụng: 1, 2

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

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 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.

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) = Ω(2^n)
  - Chứng minh bằng quy nạp P(n) ≥ 2^{n-2}
    - P(n) đúng với n ≤ 4
    - Với n ≥ 5

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

  - Vậy P(n) = Ω(2^n)

289

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 toán 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

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

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

293

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 U {x} có triển vọng là giải pháp) then S:=S U {x}
      endif
   endwhile
      if (S là lời giải) then return S
      else return 0
      endif
end
```

294

Một số ứng dụng

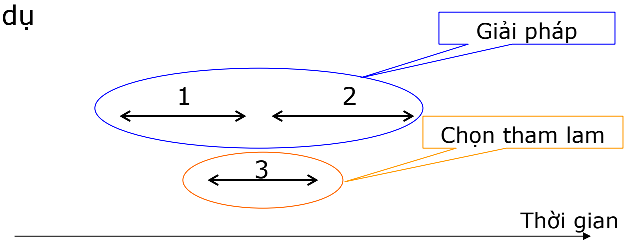
- Thuê xe
- Đổi tiền
- Xếp ba lô
- Mã Huffman
- Tìm cây khung nhỏ nhất
  - Thuật toán Kruskal
  - Thuật toán Prim
- Tìm đường đi ngắn nhất
  - Thuật toán Dijsktra

Thuê xe

- Bài toán
  - Có một chiếc xe ôtô 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 $S = \{a_1, a_2, \ldots, a_n\}$ 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 lệnh nhau về thời gian, nghĩa là yêu cầu tiếp theo chỉ được bắt đầu khi yêu cầu trước đó kết thúc
    - Hay: $\forall a_1 \in S,\ a_2 \in S$: $s(a_1) \le s(a_2) \Rightarrow f(a_1) \le f(s_2)$

Thuê xe

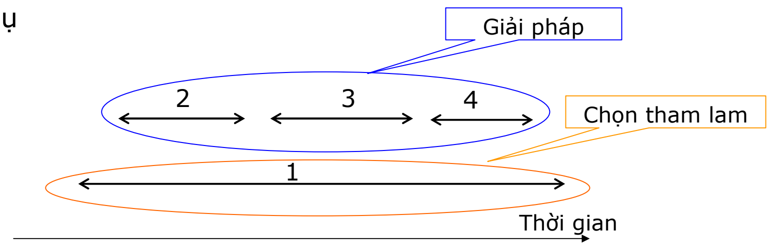
- Phép thử 1
  - Sắp xếp các yêu cầu thuê xe theo thứ tự tăng dần thời gian thuê
  - Chọn tham lam
    - chọn yêu cầu có thời gian thuê ngắn nhất
  - Ví dụ



Thuật toán không cho giải pháp tối ưu

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



Thuật toán cũng không cho giải pháp tối ưu

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 !

Thuê xe

- Chứng minh tính tối ưu của thuật toán
  - Giả sử $F=\{x_1, x_2, \ldots, x_p\}$ là giải pháp đạt được bởi thuật toán tham lam và $G=\{y_1, y_2, \ldots, y_q\}$ 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 $G'=\{y_1, y_2, \ldots, y_{k-1}, x_k, y_{k+1}, \ldots, y_q\}$ thoả mãn ràng buộc sự lệnh 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

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(nlogn)
  - 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 dường 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

303

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

304

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 S 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

    $F-\{c_1, c_2, \ldots, c_n\} = \varnothing$
  - Hay: $F = \{c_1, c_2, \ldots, c_n\}$
  - 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

305

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

306

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 !

307

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=3	S=3
50	20	20	5	1	1	1

308

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

309

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 M không chọn lựa tờ tiền mệnh giá 50. Vì M phải thoả mãn các ràng buộc trên, nên có thể chỉ có 2 khả năng: 4x1+5+10+20 < 50 và 4x1+5+2x20 <5

      Vậy nếu $S \geq 50$, F phải chứa ít nhất một tờ tiền mệnh giá 50
    - Nếu $50 \geq S \geq 20$, lý giải tương tự, F phải chứa một tờ tiền mệnh giá 20
    - Tiếp tục …

310

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'∪{p} là giải pháp tối ưu tốt hơn F cho bài toán S. Điều này mâu thuẩn giả thiết.

      Vậy F'∪{p}= F hay F'=F-{p}.

311

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
```

312

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

313

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, $0 \le x_i \le 1$, 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)

314

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

Xếp ba lô

- Thuật toán

```
xepbalotungphan()
begin
    sắp xếp các đồ vật theo tỷ lệ vi/wi giảm dần
    w = W
    i = 1
    while (w ≥ wi) do // xếp toàn bộ đồ vật vào ba lô
        xi = 1
        w = w – wi
        i = i +1
    endwhile
    xi = wi/w // đồ vật cuối cùng được chọn để xếp vào ba lô
    for k from i + 1 to n do
        xi = 0 // các đồ vật không được xếp vào bao lô
    endfor
    return (x1, x2, …,xn) // giải pháp
end
```

Xếp ba lô

- Thuật toán
  - Phân tích độ phức tạp
    - Sắp xếp: O(nlogn)
    - Lặp: duyệt n đồ vật, vậy O(n)

Xếp ba lô

- Chứng minh tính tối ưu (1)
  - Cách 1: chứng minh trực tiếp (1)
    - Giả sử $v_1/w_1 \ge v_2/w_2 \ge \ldots \ge v_n/w_n$
    - $X = \{x_1, x_2, \ldots x_n\}$ là giải pháp 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 $X=\{1,\ldots,1,x_j,0,\ldots,0\}$ (chỉ có đồ vật cuối cùng được chọn một số phần)
    - Bằng phản chứng, giả sử $Y = \{y_1, y_2, \ldots y_n\}$ giải pháp khác và V(Y) là tổng giá trị
    - Ta có

$$V(X)-V(Y)=\sum_{i=1}^{n} x_i v_i - \sum_{i=1}^{n} y_i v_i = \sum_{i=1}^{n} v_i (x_i - y_i)$$

$$= \sum_{i=1}^{n} (x_i - y_i) w_i \frac{v_i}{w_i}$$

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:

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

      Như thế: $V(X) \geq V(Y)$

      Hay V(X) là giải pháp tối ưu

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.

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

        ⇨ S phải chứa đồ vật k

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 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.

321

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

322

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

323

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 bít để biểu diễn 6 kí tự

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bít	000	001	010	011	100	101

  - Số bít cần thiết để lưu trữ tệp: 300000
  - Tồn tại giải pháp tốt hơn ?

324

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 bít	0	101	100	111	1101	1100

- Số bít cần để lưu trữ tệp

  (45x1+13x3+12x3+16x4+9x4+5x4)x1000=224000 bít
- Chính là giải pháp tối ưu cho tệp dữ liệu này

325

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 bít	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*

326

Mã Huffman

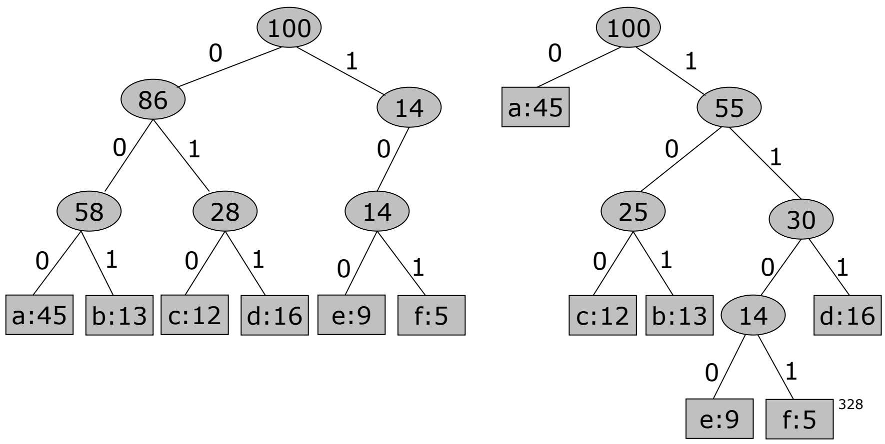
- Quá trình giải mã cần một **sự biểu diễn các mã tiền tố** để xác định mỗi mã có thể được xác định dễ dàng
  - **Cây nhị phân** với mỗi nút lá biểu diễn một kí tự
  - Mã nhị phân cho mỗi kí tự là đường đi từ nút gốc đến nút lá chứa kí tự đó
    - đọc 1 nghĩa là đi đến nút phải, đọc 0 đi đến nút trái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

Mã Huffman

- Ví dụ cây nhị phân

Mã có độ dài hằng số

Mã có độ dài thay đổi



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

329

Mã Huffman

- Tính số bít cần để mã hoá tệp
  - Giả sử T là cây nhị phân biểu diễn các mã tiền tố
  - Mỗi kí tự c ∈ 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 bít biểu diễn c
  - Số bít 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

330

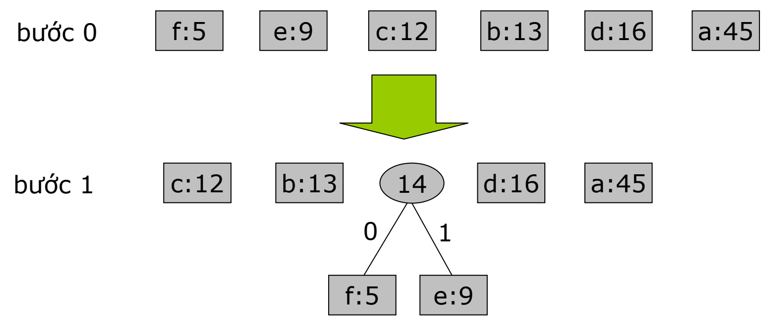
Mã Huffman

- 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

  - Cây nhị phân sẽ được xây dựng từ dưới lên
    - 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

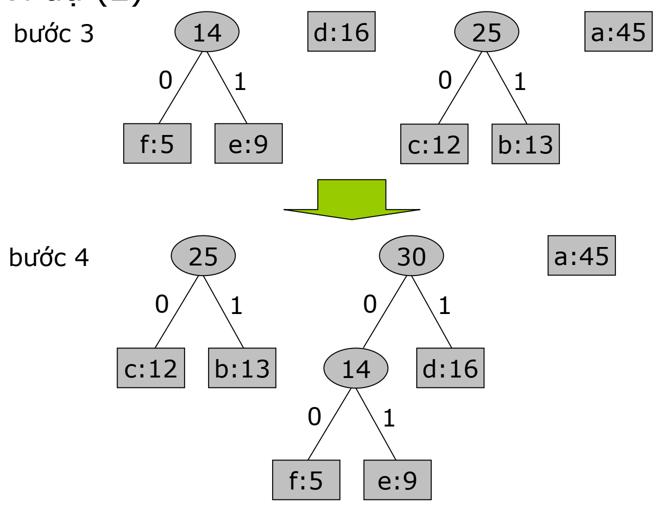
Mã Huffman

- Ví dụ (1)



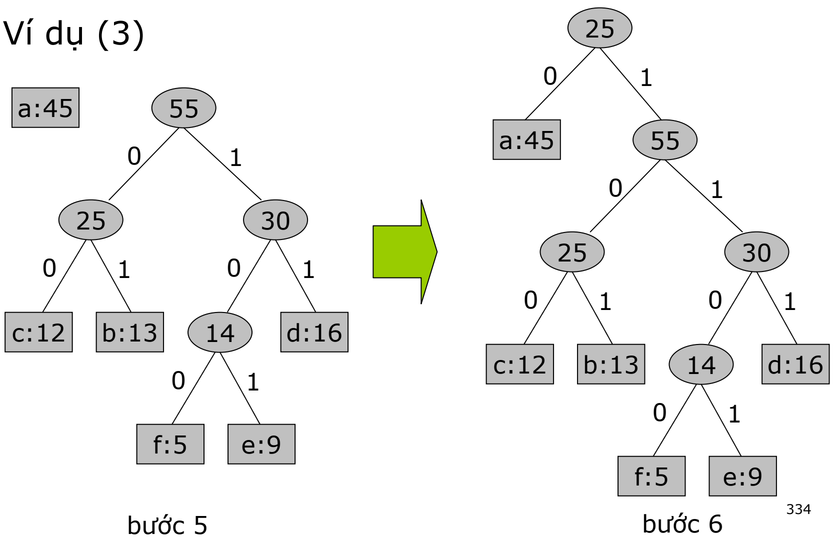
Mã Huffman

- Ví dụ (2)



Mã Huffman

- Ví dụ (3)



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 ?

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

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

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 (cạnh (u,v) ≡ cạnh (v,u))
  - Một đồ thị được gọi là *liên thông* nếu ∀u,v ∈ N 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)∈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

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***

339

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

340

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 trong 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 la cây khung nhỏ nhất

341

Thuật toán Kruskal

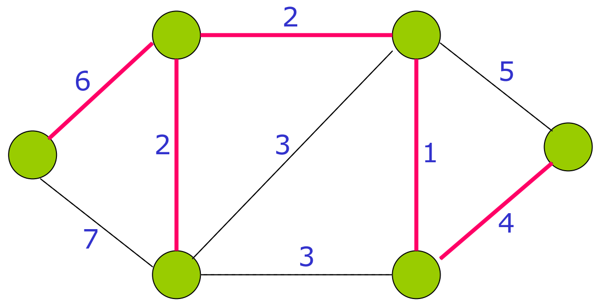
- Thuật toán

```
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
```

342

Thuật toán Kruskal

- Ví dụ



343

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

344

Thuật toán Prim

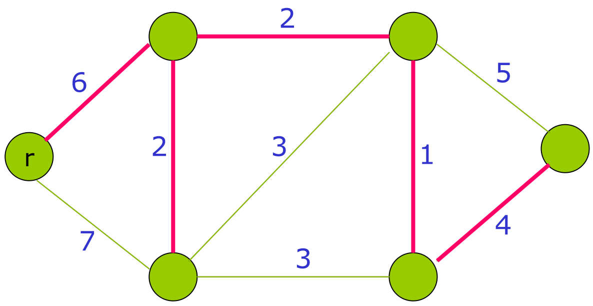
- Thuật toán

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

345

Thuật toán Prim

- Ví dụ



346

Cây khung nhỏ nhất

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

347

Đường đi ngắn nhất

- Cho đồ thị định hướng G=(V,E), E tập các cạnh
- Các cạnh 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

348

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

349

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*

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 ($m \le n$)
    (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

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)

353

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:
      (n1-1)+(n2-1) + 1 = n1+n2-1 = n-1

354

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 vét 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 vét 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 x A_2 x … x A_n (A_k là một tập hợp) sao cho mỗi phần tử s = (s_1,s_2,…,s_n) $\in$ S thoả mãn ràng buộc nào đó* »

- Ví dụ
  - Tìm tất cả các hoán vị của {1,2, …, n}

    A_k = {1,2,…,n} với $\forall$k

    $s_i \neq s_k$ với $\forall i \neq k$

Quay lui

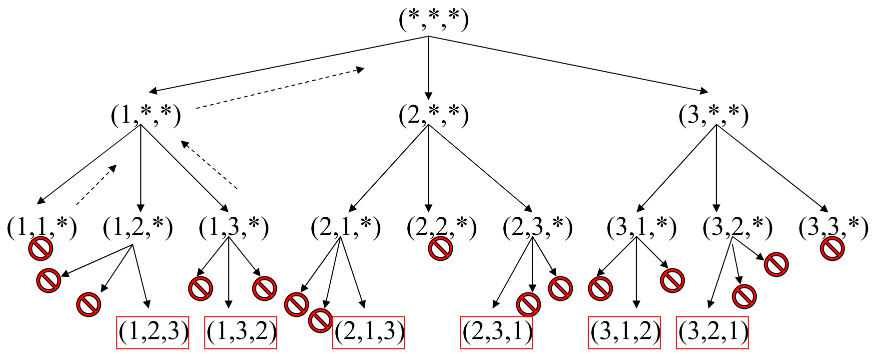
- Ý tưởng
  - Giải pháp được xây dựng từng thành phần ở mỗi bước
  - *Tìm kiếm vét cạn* tất cả các giải pháp có thể trên cây không gian trạng thái
    - Mất nhiều thời gian thực thi
  - *Tỉa bớt* các thành phần không đưa đến giải pháp
    - Chỉ những giải pháp từng phần có triển vọng được sử dụng
      - 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 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

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

Quay lui

- Ví dụ
  - Cây không gian trạng thái



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

Quay lui

- Các bước thiết kế thuật toán quay lui
  - Chọn cách *biểu diễn giải pháp*
  - Xây dựng các *tập A1, A2, …, An* và xếp thứ tự để các phần tử của chúng được xử lý
  - 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*
  - Chọn *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng* không

361

Quay lui

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

362

Quay lui

- Thuật toán quay lui tổng quát: đệ quy

quaylui-dequy (k)
begin
  if (s=(s_1, s_2, …, s_{k-1}) là giải pháp) then xuly(s)
  else
    for j from 1 to |A_k| do // thử tất cả các giá trị có thể
      s_k = a^k_j // A_k = {a^k_1, a^k_2, …}
      if ((s_1, s_2, …, s_k) là có triển vọng) then
        quaylui-dequy (k+1) // xây dựng thành phần tiếp theo
      endif
    endfor
  endif
end

363

Quay lui

- Thuật toán quay lui tổng quát: lặp

quaylui-lap(A_1, A_2, …, A_n)
begin
  k=1; i_k=0
  while (k > 0) do
    i_k=i_k+1
    v=false
    while (v=false and i_k≤|A_k|) do
      s_k=a^k_{ik}
      if ((s_1,…,s_k) là có triển vọng) then v=true
      else i_k=i_k+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
        i_k=0
      endif
    else k=k-1 endif // quay lui lại thành phần trước đó
  endwhile
end

364

Mố số ứng dụng

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

365

Hoán vị

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

366

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
```

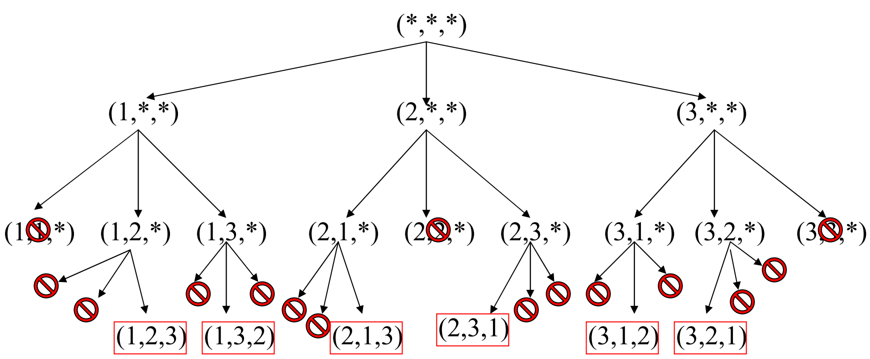
```
trienvong(s[1..k])
begin
  for i from 1 to k-1 do
    if (s[k]=s[i]) then
      return false
    endif
  endfor
  return true
end
```

sử dụng: hoanvi(1)

367

Hoán vị

- Minh hoạ: n=3



368

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, …, s_n)
  - *tập A_1, A_2, …, 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

369

Chuỗi nhị phân

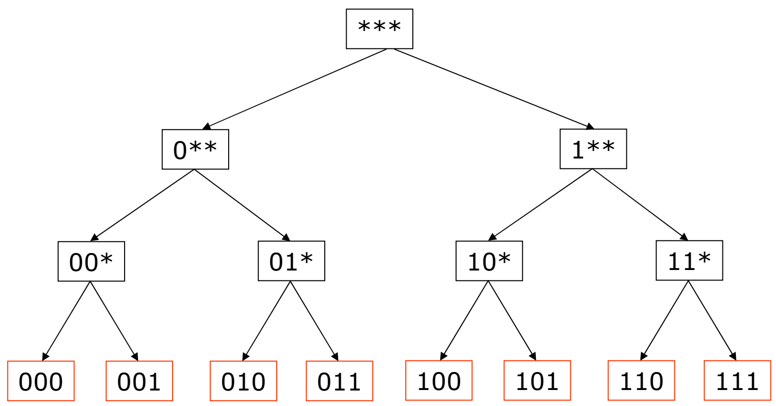
- Thuật toán

```
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
```

370

Chuỗi nhị phân

- Minh hoạ: n = 3



Chuỗi nhị phân

- Phân tích thuật toán
  - C(n) thời gian thực thi *chuoinhiphan(n)*
  - Vậy

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

  - 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 tỉa bớt các giải pháp thành phần*

Tập con

- Bài toán
  - Cho tập hợp A={a_1,a_2,…,a_n}, hãy sinh ra tất cả các tập con của A
  - Ví dụ: A = {1,2,3}, m =2, S={{1,2},{1,3},{2,3}}
- Các bước thiết kế
  - *biểu diễn giải pháp*
    - mỗi tập hợp con được biểu diễn bởi một véc-tơ s=(s_1, s_2, …, s_n), nếu s_i = 1 thì a_i thuộc s, s_i = 0 thì ngược lại
  - *tập A_1, A_2, …, 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, …, 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

Tập con

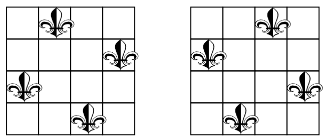
- Thuật toán

```
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 ?
  - Chỉnh sửa thuật toán để tạo ra tất cả các tập hợp con của A có đúng m phần tử (m<n)

Xếp n con hậu

- Bài toán
  - Tìm tất cả các khả năng xếp n con hậu trên một bàn cờ có nxn 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



375

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 $s=(s_1, s_2, \ldots, s_n)$ với s_k = *cột mà con hậu k được đặt trên đó*
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1,\ldots,n\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả 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

376

Xếp n con hậu

Điều kiện tiếp tục (1)

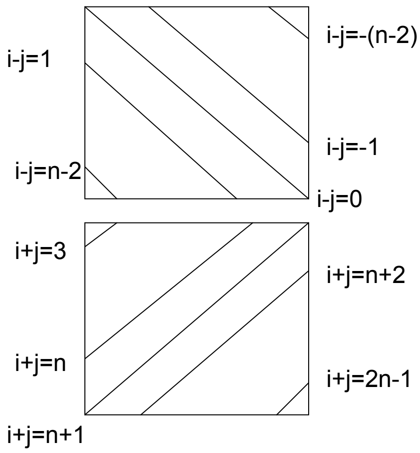
- giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn ràng buộc bài toán (mỗi hàng/cột/đường chéo chỉ chứa đúng một con hậu)
- mỗi hàng chứa đúng một con hậu: điều này luôn đúng do cách biểu diễn giải pháp
- mỗi cột chứa đúng một con hậu: $s_i \neq s_j$ với mọi $i \neq j$. Chỉ cần kiểm tra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

377

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: $|j-i| = |s_j-s_i|$



378

Xếp n con hậu

- Thuật toán

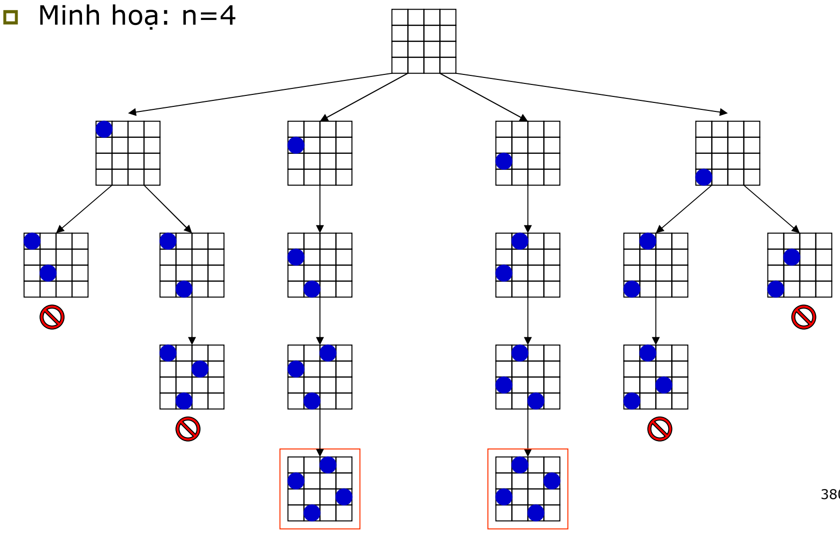
```
xephau (k)
begin
   if (k=n+1) then
      print(s[1..n])
   else
      for i from 1 to n do
         s[k]=i
         if (trienvong(s[1..k])) then
            xephau(k+1)
         endif
      endfor
   endif
end
```

```
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
```

379

Xếp n con hậu

- Minh hoạ: n=4



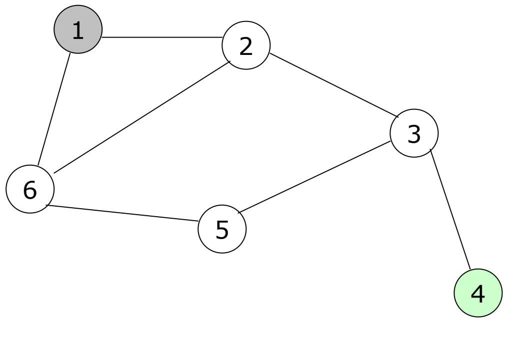
380

Xếp n con hậu

- Nhận xét
  - Với n=4
    - Nếu sử dụng kỹ thuật *tỉa bớt* chỉ có 2 giải pháp cuối cùng được xét
    - Nếu không sử dụng kỹ thuật tỉa bớt, 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 tỉa bớt của thuật toán quay lui tiết kiệm rất lớn thời gian thực thi

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

$1 \to 2 \to 3 \to 4$

$1 \to 2 \to 6 \to 5 \to 3 \to 4$

$1 \to 6 \to 2 \to 3 \to 4$

$1 \to 6 \to 5 \to 3 \to 4$

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]

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

  - Tìm tất cả các đường đi $s=(s_1,s_2,\ldots,s_m)$, s_k trong {1,…,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)

383

Tìm đường đi

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \ldots, s_m)$ với s_k *thành phố đi đến ở bước k*
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1,\ldots,n\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn:
      - $s_i \neq s_k$ với mọi i trong {1,…,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 đính

384

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
```

```
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
```

s_1 = thành phố xuất phát
duongdi(2)

385

Xếp ba lô 0-1

- Bài toán
  - Có n đồ vật có trọng lượng $w_1,\ldots,w_n$ và giá trị tương ứng $v_1, \ldots, v_n$. 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**

386

Xếp ba lô 0-1

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \ldots, s_n)$ với nếu s_i = 1 thì đồ vật i được chọn, s_i = 0 ngược lại thì không
  - *tập $A_1, A_2, \ldots, A_n$ 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*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn:
      $k \leq n$
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - k = n
      $\sum_{i=1}^{n} s_i w_i = W$
    - So sánh giải pháp với giải pháp trước đó, lưu lại giải pháp có tổng giá trị lớn nhất

387

Xếp ba lô 0-1

- Thuật toán

giatri-totnhat được khởi gán bằng 0

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

Chọn giải pháp tốt hơn

388

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, …, s_k) phải thoả mãn:

      k≤n

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

389

Xếp ba lô 0-1

- Cải tiến thuật toán (2)

```
xepbalo (k)
begin
   if (k-1 = n) then
      giatri = ∑_{i=1}^{n} s_i v_i
      if (giatri > giatri-totnhat) then
          giatri-totnhat=giatri
          giaphap-totnhat={s_1,s_2,…,s_n}
      endif
   else
      s[k] = 0; xepbalo (k+1)
      if (∑_{i=1}^{k} s_i w_i ≤ W) then
          s[k] = 1; xepbalo (k+1)
      endif
   endif
end
```

Tỉa bớt
Có thể thay bằng hàm triển vọng

390

Xếp ba lô 0-1

- Thuật toán đơn giản hơn

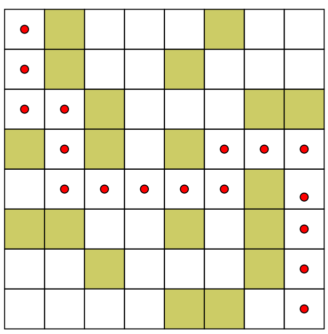
```
xepbalo (k, W)
begin
   if (k-1 = n) then
      giatri = Σ_{i=1}^{n} s_i v_i
      if (giatri > giatri-totnhat) then
          giatri-totnhat=giatri
          giaphap-totnhat={s_1,s_2,…,s_n}
      endif
   else
      s[k] = 0; xepbalo (k+1, W)
      if (W ≥ w_k) then
          s[k] = 1; xepbalo (k+1, W-w_k)
      endif
   endif
end
```

391

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

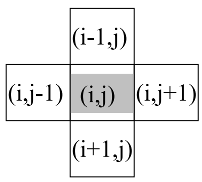
Mê cung

- Bài toán
  - Giả sử một mê cung được định nghĩa là một lưới có nxn 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)



392

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 $s=(s_1,s_2,\ldots,s_m)$, s_k trong {1,…,n}x{1,…,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

393

Mê cung

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \ldots, s_m)$ với s_k *ô đi đến ở bước k*
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - A_k = {1,…,n}x{1,…,n}, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn:

      $s_k \neq s_q$ với mọi q trong {1,…,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)

394

Mê cung

- Thuật toán (1)

```
mecung (k)
begin
  if (s[k-1]=(n,n)) then print (s[1..k])
  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
```

395

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 bàn cờ
  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)

396

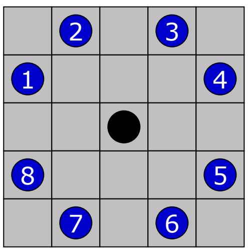
Bài tập

- 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 ≥ 4 màu sao cho hai nước láng giềng bất kỳ có màu khác nhau

Bài tập

- Ngựa đi tuần
  - Xuất phát từ một ô bất kỳ trên bàn cờ 8x8 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

Thuật toán xác xuất (9)

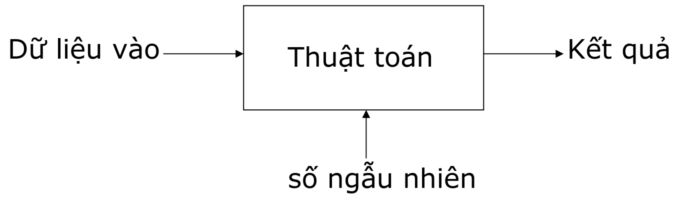
Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Thuật toán đơn định (deterministic algorithm)

Dữ liệu vào → Thuật toán → Kết quả

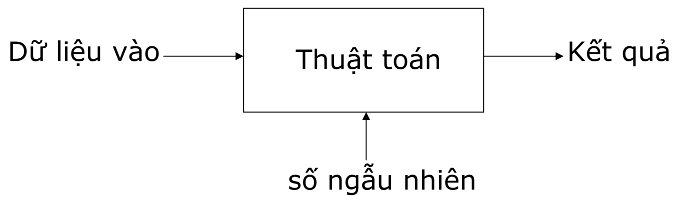
- Mục đích: chỉ ra rằng thuật toán thực hiện *đúng đắn* và *hiệu quả* (điển hình, số bước thực hiện là hàm đa thức theo kích thước dữ liệu vào)
- Kết quả của thuật toán chỉ phụ thuộc vào dữ liệu vào
- Chúng ta chỉ mới xem xét các thuật toán đơn định

Thuật toán xác suất (probabiliste/randomized algorithm)



- Kết quả của thuật toán chỉ phụ thuộc vào dữ liệu vào mà còn phụ thuộc vào một số ngẫu nhiên
  - Tạo nên sự lựa chọn ngẫu nhiên trong khi thực thi thuật toán
- Hoạt động thuật toán có thể khác nhau trên cùng một dữ liệu vào
- Thuật toán xác suất còn được gọi là thuật toán ngẫu nhiên

Thuật toán xác suất (probabiliste/randomized algorithm)



- Thiết kế thuật toán + phân tích thuật toán *dường như* (với một xác suất cao) hiệu quả trên *mọi dữ liệu vào*
- Xác suất hiệu quả của thuật toán dựa trên số ngẫu nhiên

Phân biệt với phân tích xác suất (probabiliste analysis of algorithm)

- Dữ liệu vào được chọn lựa một cách ngẫu nhiên (với một xác suất phân bố)
- Phân tích hoạt động thuật toán trong trường hợp trung bình
- Chỉ ra rằng thuật toán hoạt động trên hầu hết các dữ liệu vào

Các loại thuật toán xác suất

- Phân biệt hai thuật toán xác suất cơ bản
  - Monte Carlo
    - thực thi một số bước xác định đối với mỗi dữ liệu vào
    - cho một kết quả mà khả năng đúng với một xác suất xác định
      - có thể cho kết quả sai
  - Las Vegas
    - luôn cho một kết quả đúng
    - thời gian thực thi đối với mỗi dữ liệu vào là một biến ngẫu nhiên mà có khả năng được giới hạn
      - có thể không dừng

Monte Carlo và Las Vegas

- Yếu tố xác suất trong thuật toán xác suất chỉ phụ thuộc vào các lựa chọn ngẫu nhiên được thực hiện bởi thuật toán
  - độc lập với dữ liệu vào

Thuật toán	Dừng ?	Đúng ?
Monte Carlo	Có	
Las Vegas		Có

405

Ưu và nhược điểm của thuật toán xác suất

- Ưu điểm
  - Đơn giản
  - Hiệu quả
- Nhược điểm
  - Hai lần thực thi một thuật toán xác suất trên cùng một dữ liệu vào có thể cho kết quả khác nhau
  - Với một **xác suất rất thấp**
    - Có thể cho kết quả sai
    - Có thể không dừng

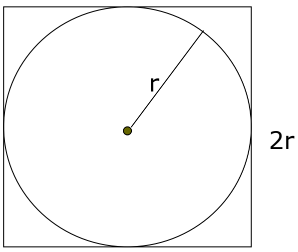
406

Một số thuật toán Monte Carlo

- Tính π
- Tính tích phân
- Kiểm tra tích hai ma trận
- Nhát cắt cực tiểu (minimum cut)

Tính π

- Tính gần đúng số π



  - Chọn một điểm p trong hình vuông một cách ngẫu nhiên và đồng đều
  - Câu hỏi: xác suất để điểm p nằm trong hình tròn bằng bao nhiêu ?
  - Trả lời: $\pi/4$

Tính π

- Ý tưởng
  - chọn một số lớn *n* điểm trong hình vuông một cách ngẫu nhiên, đồng đều và độc lập
  - giả sử có *k* điểm nằm trong hình tròn
  - Vậy $k/n \approx \pi/4$
  - Hay $\pi \approx 4k/n$

Tính π

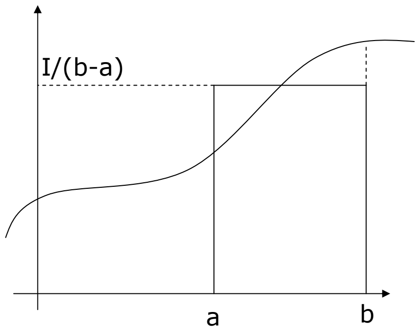
- Thuật toán

```
tinh-gan-dung-π (n)
begin
   k = 0
   for i from 1 to n do
      x = random (0,1)
      y = random (0,1)
      if (x² + y² ≤ 1) then
         k = k + 1
      endif
   endfor
   return (4*k/n)
end
```

Tính tích phân

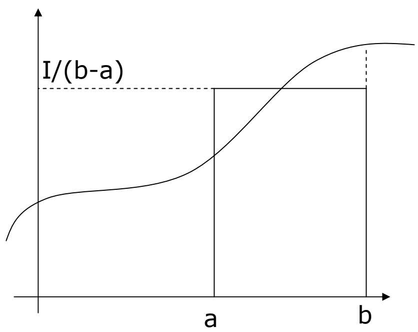
- Cho f : R → $R^{\geq 0}$ là một hàm liên tục và a,b ∈ R sao cho a ≤ b, diện tích dưới hàm y=f(x) giữa x=a và x=b được định nghĩa:

$$I = \int_a^b f(x)dx$$



Tính tích phân

- **Câu hỏi**: tích phân I có thể được tính dễ dàng ?
- **Trả lời**: phụ thuộc vào hàm f(x)
- Nếu f(x) không đơn giản để tính tích phân, chúng ta sử dụng thuật toán xác suất
- Hình chữ nhật có chiều cao I/(b-a) giữa x=a và x=b có diện tích xấp xỉ diện tích dưới hàm y=f(x) giữa x=a và x=b
  - "Chiều cao" trung bình của đoạn đường cong f(x) giữa x=a và x=b cũng là I/(b-a)



Tính tích phân

- Ý tưởng
  - Tính "chiều cao" trung bình *h* của đoạn đường cong
  - Vậy diện tích *I = h*(b-a)*
- Thuật toán

```
tinh-tich-phan (n)
begin
   s = 0
   for i from 1 to n do
      x = random (a,b)
      s = s + f(x)
   endfor
   h = s/n
   return (h*(b-a))
end
```

413

Tính tích phân

- Xác suất lỗi của giá trị tích phân tính bởi thuật toán được chứng minh là $1/\sqrt{n}$
- Tồn tại phiên bản đơn định của thuật toán này tính tích phân với kết quả tương đương
  - chọn các điểm với khoảng cách bằng nhau
- Tuy nhiên, phiên bản đơn định chỉ hiệu quả trong trường hợp tính tích phân 1 lớp, rất phức tạp (thời gian) trong trường hợp tính tích phân nhiều lớp (>3)

414

Kiểm tra tích ma trận

- Bài toán
  - Giả sử *A*, *B* và *C* là các ma trận có kích thước *nxn* với các phần tử có giá trị trong tập hợp *F*. Cần kiểm tra A.B = C ?
  - Thuật toán đơn giản
    - Độ phức tạp O(n^3)
  - Thuật toán Strassen
    - Độ phức tạp O($n^{\log 7}$) = O($n^{2.38}$)

415

Kiểm tra tích ma trận

- Thuật toán xác suất (1)

```
kiemtratichmatran (A, B, C)
// kiểm tra tích A.B = C ?
begin
   Tạo một véc-tơ x có n phần tử lấy ngẫu nhiên từ tập F
   Tính y = A.(B.x)
   if (y = C.x) then
      return (true)
   else
      return (false)
   endif
end
```

416

Kiểm tra tích ma trận

- Thuật toán xác suất (2)
  - Độ phức tạp $O(n^2)$
  - Nếu AB = C thì thuật toán luôn cho kết quả đúng
  - Nếu AB ≠ C thì thuật toán cho kết quả sai (nghĩa là AB = C) với xác xuất 1/|F|

417

Kiểm tra tích ma trận

- Giảm xác suất lỗi xảy ra
  - Lặp việc thuật toán kiểm tra *k* lần, với mỗi lần véc-tơ *r* được chọn một cách ngẫu nhiên và độc lập từ F
  - Khi đó
    - nếu có vài lần kiểm tra cho kết quả sai, thì chắc chắn AB≠C
      - bởi vì nếu AB=C thì tất cả các lần kiểm tra đều cho kết quả đúng
    - nếu AB≠C mà tất cả các lần kiểm tra đều cho kết quả đúng với xác xuất $(1/|F|)^k = |F|^{-k}$
  - Độ phức tạp $O(kn^2)$

418

Nhát cắt cực tiểu (minimum cut)

- Bài toán
  - Cho đồ thị vô hướng liên thông G=(V,E), V là n tập đỉnh, E là tập m cạnh
  - *Nhát cắt* là tập các cạnh bị loại bỏ sao cho đồ thị thu được ít nhất là hai thành phần liên thông tách rời V_1 và V_2: $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$
  - *Nhát cắt cực tiểu* là nhát cắt có kích thước nhỏ nhất
    - Nhát cắt cực tiểu có kích thước tối đa bằng bậc của đỉnh nhỏ nhất trong đồ thị
  - Ví dụ ứng dụng
    - Giả sử mạng máy tính được biểu diễn bằng đồ thị, khả năng « bền vững » của mạng được xác định bởi số kết nối tối thiểu khi bị hỏng làm ngắt mạng

419

Nhát cắt cực tiểu

- Thuật toán xác suất

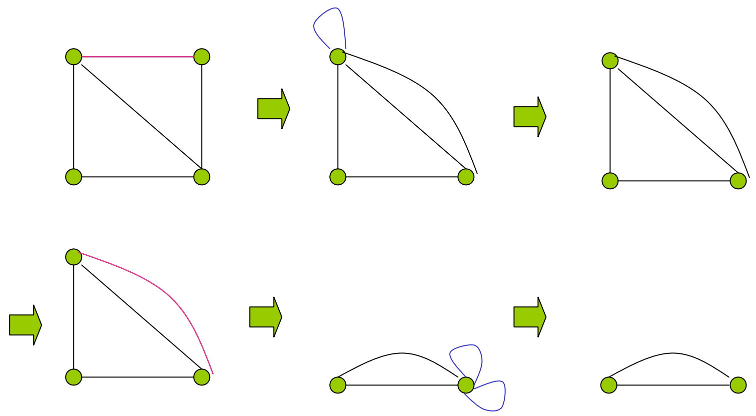
```
nhatcatcuctieu (G = (V,E))
begin
   while (|V| ≥ 2) do
       Chọn ngẫu nhiên một cạnh e = (u,v) của G
       Rút gọn u và v thành một đỉnh
       Loại bỏ tất cả các cạnh nối u và v
   endwhile
   return tất cả các cạnh E của G
end
```

  - Nhận xét
    - Nhát cắt của đồ thị G cũng chính là nhát cắt của đồ thị G sau khi đã rút gọn đỉnh u và đỉnh v
    - Khi đồ thị chỉ còn 2 đỉnh, thì nhát cắt chính là tất cả các cạnh nối u và v
  - Độ phức tạp: $O(n^2)$

420

Nhát cắt cực tiểu

- Minh hoạ



Nhát cắt cực tiểu

- Phân tích thuật toán (1)
  - *Câu hỏi*: thuật toán luôn luôn cho nhát cắt cực tiểu ?
  - *Trả lời*: thuật toán cho nhát cắt cực tiểu với một xác suất ≥ $2/n^2$ (n số đỉnh)
  - Chứng minh ?

Nhát cắt cực tiểu

- Phân tích thuật toán (2)
  - Gọi C là nhát cắt cực tiểu, giả sử |C| = k
  - Mỗi đỉnh phải có bậc ≥ k
    - Nếu tồn tại đỉnh có bậc < k thì cắt ngay tại đỉnh đó, khi đó C không phải là nhát cắt cực tiểu (mâu thuẫn)
  - Nghĩa là mỗi đỉnh có ít nhất *k* cạnh nối đến nó
  - Vì đồ thị có n đỉnh, nên có ít nhất *kn* cạnh (có thể trùng nhau)
  - Tuy nhiên, mỗi cạnh nối 2 đỉnh, nên đồ thị có ít nhất *kn/2* cạnh

423

Nhát cắt cực tiểu

- Phân tích thuật toán (3)
  - Thuật toán lặp n-2 bước
    - Bước lặp 1
      - Xác suất để một cạnh được chọn thuộc C là ≤ $\frac{k}{kn/2} = 2/n$
      - Xác suất để một cạnh được chọn không thuộc C là ≥ *1-2/n*
    - Bước lặp 2 (chỉ còn n-1 đỉnh)
      - Xác suất để một cạnh được chọn thuộc C là ≤ *2/(n-1)*
      - Xác suất để một cạnh được chọn không thuộc C là ≥*1-2/(n-1)*
    - …
    - Bước lặp n-2
      - Xác suất để một cạnh được chọn thuộc C là ≤ *2/3*
      - Xác suất để một cạnh được chọn không thuộc C là ≥ *1-2/3*

424

Nhát cắt cực tiểu

- Phân tích thuật toán (4)
  - Xác suất *p* để thuật toán thành công (cho nhát cắt cực tiểu C) bằng xác suất tất cả các cạnh được chọn lựa không thuộc C trong mỗi bước lặp
    - khi đó các cạnh còn lại chính là C
  - Nghĩa là

$$
\begin{aligned}
p &\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{3-2}{3}\right) \\
&= \frac{2}{n(n-1)} \geq \frac{2}{n^2}
\end{aligned}
$$

Nhát cắt cực tiểu

- Nâng cao xác suất thành công
  - Thực hiện lặp thuật toán *nhatcatcuctieu* $n^2/2$ lần

```
nhatcatcuctieu' (G)
begin
    min = ∞, C = ∅
    for i from 1 to n²/2 do
        C = nhatcatcuctieu (G)
        if (min > |C|) then  min = |C| endif
    endfor
    return (C)
end
```

  - Xác suất thuật toán không xác định được nhát cắt cực tiểu là

$$(1-2/n^2)^{n^2/2} \cong 1/e$$

    - Được tính từ công thức $\lim_{x\to\infty}\left(1-\frac{2}{x}\right)^x \cong \frac{1}{e}$
  - Nghĩa là xác suất thuật toán thành công ≥ 60%

Một số thuật toán Las Vegas

- Quicksort
- Chọn phần tử

427

Quicksort

- Thuật toán đơn định Quicksort
  - Trường hợp trung bình: O(nlogn)
  - Trường hợp xấu nhất: $O(n^2)$
    - Khi mảng đã được sắp xếp theo chiều ngược lại
- Thuật toán xác suất Las Vegas
  - Trường hợp xấu nhất: O(nlogn)

428

Quicksort

- Thuật toán xác suất

```
prob-quicksort (A)
begin
  if (n = 1) then return (A)
  else
        Chọn ngẫu nhiên một phần tử x trong danh sách A
        Chia danh sách A thành A1, A2, A3 sao cho các phần tử
        của A1 nhỏ hơn x, các phần tử của A2 bằng x và các phần
        tử của A3 lớn hơn x
        return (prob-quicksort(A1), A2, prob-quicksort(A3))
  endif
end
```

429

Quicksort

- Đánh giá độ phức tạp (1)
  - Độ phức tạp (số lượng phép so sánh)
    - không chỉ phụ thuộc dữ liệu (mảng cần sắp xếp)
    - còn phụ thuộc vào sự lựa chọn ngẫu nhiên phần tử chốt x
  - Giả sử $a_1 \leq a_2 \leq \ldots \leq a_n$ là n phần tử của A
  - $1 \leq i < j \leq n$, gọi X_{ij} là *biến ngẫu nhiên*

    $X_{ij} = 1$ nếu a_i được so sánh với a_j

    $X_{ij} = 0$ ngược lại
  - Khi đó, số phép so sánh được thực hiện bởi thuật toán là:

$$X = \sum_{1 \leq i \leq j \leq n} X_{ij}$$

430

Quicksort

- Đánh giá độ phức tạp (2)
  - Độ phức tạp của thuật toán chính là *giá trị kỳ vọng* của biến ngẫu nhiên X

$$E(X) = E\left(\sum_{1\le i<j\le n} X_{ij}\right) = \sum_{1\le i<j\le n} E(X_{ij}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{ij})$$

  - Mà: $E(X_{ij}) = 0.Pr(X_{ij}=0) + 1.Pr(X_{ij}=1) = Pr(X_{ij}=1)$
  - $Pr(X_{ij} = 1) \equiv$ xác suất a_i được so sánh với a_j
  - Để a_i được so sánh với a_j, thì hoặc a_i hoặc a_j phải được chọn làm chốt trước bất kỳ phần tử nào từ $a_{i-1},\ldots,a_{j-1}$
    - bởi vì nếu một trong những phần tử $a_{i-1},\ldots,a_{j-1}$ đượck chọn là chốt thì a_i và a_j sẽ thuộc hai mảng con khác nhau

431

Quicksort

- Đánh giá độ phức tạp (3)
  - Xác suất a_i hoặc a_j được chọn làm chốt = xác xuất phần tử bất kỳ từ $a_i,\ldots,a_j$ (j-i+1 phần tử) được chọn làm chốt
  - Nghĩa là $P(X_{ij}=1) = 2/(j-i+1)$
  - Vậy

$$E(X) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}$$

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

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

$$= 2n\sum_{k=1}^{n}\frac{1}{k}$$

432

Quicksort

- Đánh giá độ phức tạp (4)
  - Mà dãy 1+1/2+1/3 + … + 1/n ≈ ln(n)
  - Vậy E(X) ≤ 2nln(n)

  - Như thế, độ phức tạp của thuật toán prob-quicksort là O(nln(n))

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(nlog(n))*, độ phức tạp bài toán chọn phần tử cũng là *O(nlog(n))*
- Thuật toán chia để trị
  - Đề xuất bởi M. R. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan
  - Độ phức tạp trong trường hợp trng bình là O(n)
  - Đã trình bày trong các thuật toán chia để trị
- Tồn tại thuật toán xác suất đơn giản
  - Độ phức tạp trong trường hợp trong bình cũng là O(n)

Chọn phần tử

- Thuật toán xác suất

```
chonphantu (A, k)
begin
   if (n = 1) then return (A[1])
   else
         Chọn ngẫu nhiên x ∈ A
         Chia A thành A1, A2 sao cho A1= {y ∈ A| y <x} và
         A2 ={y ∈ A| x < y}
         Giả sử |A1| = j (nghĩa là, x là phần tử lớn thứ j+1)
         if (k = j+1) then return (x)
         else if (k ≤ j) then return (chonphantu (A, k))
                else return (chonphantu (A, k-j-1))
                endif
         endif
   endif
end
```

435

Chọn phần tử

- Phân tích thuật toán (1)
  - Gọi C(n) là độ phức tạp chọn phần tử thứ k trong danh sách n phần tử
  - C(1) = 0
  - Có $|A_1| = j$ và $|A_2| = n-j-1$
  - Khi đó, thời gian trung bình cần cho các lời gọi đệ quy là

    C(j) và C(n-j-1)

    j có thể có giá trị từ 0 đến n-1 với xác suất 1/n
  - Nghĩa là, thời gian trung bình cho các lời gọi đệ quy nhiều nhất là

$$\frac{1}{n}\sum_{j=0}^{n-1}\left(C(j)\ hay\ C(n-j-1)\right)$$

436

Chọn phần tử

- Phân tích thuật toán (2)

$$\frac{1}{n}\sum_{j=0}^{n-1}\left(C(j)\ hay\ C(n-j-1)\right)$$

  - Khi nào thì C(j), khi nào thì C(n-j-1)
    - Khi $k \le j$, đệ quy trên S_1, nghĩa là C(j)
    - Khi k = j + 1, kết thúc, nghĩa là 0
    - Khi k > j + 1, đệ quy trên S_2, nghĩa là C(n-j-1)
  - Vậy, thời gian trung bình cho các lời gọi đệ quy nhiều nhất là

$$\frac{1}{n}\left(\sum_{j=0}^{k-2}C(n-j-1)+\sum_{j=k}^{n-1}C(j)\right)$$

437

Chọn phần tử

- Phân tích thuật toán (3)
  - Chia A thành A_1 và A_2 cần n-1 phép so sánh
  - Vậy

$$C(n) \le \frac{1}{n}\left(\sum_{j=0}^{k-2}C(n-j-1)+\sum_{j=k}^{n-1}C(j)\right)+n-1$$

$$\le \frac{1}{n}\left(\sum_{j=n-1}^{n-k+1}C(j)+\sum_{j=k}^{n-1}C(j)\right)+n-1 \qquad (1)$$

  - Thay vì tính C(n), chúng ta chứng minh bằng quy nạp rằng $C(n) \le cn$ với c là một hằng số nào đó
    - Bước cơ bản: rỏ ràng đúng khi n=1
    - Bước giả thiết: giả sử $C(m) \le cm$ với mọi $m \le n-1$
    - Bước quy nạp: chứng minh $C(n) \le cn$

438

Chọn phần tử

- Phân tích thuật toán (4)
  - Bước quy nạp: chứng minh C(n) ≤ cn
  - Thay giả thiết quy nạp vào (1), ta có

$$C(n) \le \frac{c}{n}\left(\sum_{j=n-1}^{n-k+1} j + \sum_{j=k}^{n-1} j\right) + n - 1 \qquad (2)$$

  - Mà
$$\sum_{j=n-k+1}^{n-1} j + \sum_{j=k}^{n-1} j = \sum_{j=1}^{n-1} j - \sum_{j=1}^{n-k+1-1} j + \sum_{j=1}^{n-1} j - \sum_{j=1}^{k-1} j$$
$$= 2\frac{(n-1)n}{2} - \frac{(n-k)(n-k+1)}{2} - \frac{(k-1)k}{2}$$
$$= \frac{2n^2 - 2n - n^2 + 2nk - k^2 - n + k - k^2 + k}{2}$$
$$= \frac{n^2 + 2nk - 2k^2 - 3n + 2k}{2} \le \frac{n^2 + 2nk - 2k^2}{2}$$

439

Chọn phần tử

- Phân tích thuật toán (5)
  - Mà $2nk-2k^2 \le n^2/2$
  - Vậy
$$\sum_{j=n-1}^{n-k+1} j + \sum_{j=k}^{n-1} j \le \frac{3n^2}{4}$$
  - Thay vào (2) ta có
$$C(n) \le \frac{c}{n}\frac{3n^2}{4} + n - 1 = \frac{3cn}{4} + n - 1 \le n\left(\frac{3c}{4} + 1\right)$$
  - Cuối cùng ta chứng minh được C(n) ≤ cn với c ≥ 4
  - Vậy, thuật toán có độ phức tạp trong trường hợp trung bình O(n)

440

Giá trị kỳ vọng

- **Giá trị kỳ vọng**, **giá trị mong đợi** (hoặc **kỳ vọng toán học**) của một biến ngẫu nhiên là tổng xác suất của mỗi kết quả có thể của thử nghiệm nhân với giá trị của kết quả đó
- Như vậy, nó biểu diễn giá trị trung bình mà người ta "mong đợi" thắng cược nếu đặt cược liên tục nhiều lần với khả năng thắng cược là như nhau
- Nếu X là một biến ngẫu nhiên rời rạc với các giá trị x_1, x_2, ... và các xác suất tương ứng là p_1, p_2, ... với tổng bằng 1, thì E(X) có thể được tính bằng tổng của chuỗi

$$E(X) = \sum_i x_i p_i$$

441

Lớp các bài toán NP đầy đủ (10)

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

Đặt vấn đề

- Phần lớn các thuật toán đã trình bày có độ phức tạp hàm đa thức
  - *Câu hỏi*
    - Tất cả các bài toán đều được giải quyết bởi thuật toán với thời gian thực thi hàm đa thức ?
  - *Trả lời*
    - Không. Có những bài toán không thể giải quyết được (tính dừng không xác định)
    - Không. Bởi vì có những bài toán chúng ta chỉ biết các thuật toán giải quyết chúng với thời gian thực thi hàm mũ
- Các lớp độ phức tạp
  - Lớp các vấn đề có thể giải quyết bởi thuật toán có độ phức tạp hàm đa thức
    - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Lớp các vấn đề phức tạp hơn (giải quyết bởi thuật toán có độ phức tạp hàm mũ)
    - $O(2^n)$, $O(n^n)$, $O(n!)$
- Cần phương tiện xác định một bài toán thuộc lớp nào ?

443

Bài toán

- Bài toán quyết định
  - Trong lý thuyết NP-đầy đủ, chỉ giới hạn giải quyết các **bài toán quyết định**
    - các bài toán cần giải pháp là câu trả lời hoặc *đúng* hoặc *sai*
      - ví dụ, bài toán kiểm tra một số có là số nguyên tố không ?
- Bài toán tối ưu
  - Phần lớn các bài toán cần giải quyết không là các bài quyết định, mà là các bài toán tối ưu (tìm giải pháp tốt nhất)
    - Ví dụ, bài toán xếp ba lô
- Chuyển các bài toán tối ưu về các bài toán quyết định
  - Áp đặt một cận lên giá trị cần tối ưu

444

Bài toán

- Ví dụ
  - Chuyển các bài toán tối ưu về các bài toán quyết định
    - Bài toán tối ưu: « cho đồ thị G tìm đường đi ngắn nhất giữa hai đỉnh u và v của đồ thị »
    - Bài toán quyết định: « cho đồ thị G, hai đỉnh u và v, và một số nguyên dương k, có tồn tại trong G đường đi từ u đến v có chiều dài lớn nhất là k ? »

445

Lớp P

- Định nghĩa
  - **Lớp độ phức tạp P** (*polynomial time*) được định nghĩa gồm các bài toán quyết định được giải quyết bởi thời gian đa thức
    - nghĩa là tồn tại các thuật toán có độ phức tạp hàm đa thức giải quyết bài toán
- Phần lớn các bài toán chúng ta đã xem xét là bài toán P

446

Lớp NP

- Định nghĩa
  - **Lớp độ phức tạp NP** (*nondeterministic polynomial time*) bao gồm tập hợp các bài toán quyết định được mà tồn tại thuật toán có độ phức tạp hàm đa thức kiểm tra giải pháp là đúng
    - Chúng ta không tìm cách xác định giải pháp mà chỉ kiểm tra một giải pháp đã cho là đúng

447

P và NP

- Lớp P
  - Tập hợp các bài toán có thể **được giải quyết** bởi thuật toán có độ phức tạp hàm đa thức
  - Các giải pháp **dễ dàng được tìm thấy**
- Lớp NP
  - Tập hợp các bài toán mà giải pháp của chúng có thể **được kiểm tra** bởi thuật toán có độ phức tạp hàm đa thức
  - các giải pháp **dễ dàng được kiểm tra**, cho dù rất khó khăn để tìm ra giải pháp
- $P \subseteq NP$
- P = NP ?
  - Vấn đề chưa giải quyết được !

448

P và NP

- Ví dụ
  - Bài toán tô màu đồ thị (bản đồ)
    - Cho đồ thị G và k màu. Có thể tô màu đồ thị G với chỉ nhiều nhất k màu sao cho hai đỉnh kề nhau bất kỳ có màu khác nhau ?
  - Không thuộc lớp các bài toán P
    - Để xác định giải pháp là không đơn giản
  - Thuộc lớp các bài toán NP
    - Giả sử có giải pháp là danh sách các màu được gán cho mỗi đỉnh.
    - Thuật toán kiểm tra tính đúng đắn của giải pháp là rất đơn giản:
      1. Kiểm tra mỗi đỉnh phải có màu hợp lệ (chỉ sử dụng k màu)
      2. Kiểm tra mỗi cạnh bất kỳ có hai đỉnh được tô hai màu khác nhau

Lớp NP-đầy đủ

- Các bài toán NP-đầy đủ là các bài toán « khó nhất » trong lớp NP
  - Nếu *một* bài toán NP-đầy đủ bất kỳ được giải quyết bởi thuật toán có độ phức tạp hàm đa thức
  - Thì *tất cả* các bài toán NP đều được giải quyết bởi thuật toán có độ phức tạp hàm đa thức
  - Khi đó P = NP
  - Tuy nhiên, thực tế *chưa có* một bài toán NP-đầy đủ nào được giải quyết bởi thuật toán có độ phức tạp hàm đa thức
  - Hiểu NP-đầy đủ
    - Khái niệm « rút gọn bài toán »

Rút gọn bài toán

- Một bài toán P có thể được rút gọn thành bài toán Q, nếu mỗi bài toán cụ thể (tuỳ theo dữ liệu vào) x của P có thể được *chuyển* thành mỗi bài toán cụ thể y của Q
- Khi đó, giải quyết Q(y) sẽ cung cấp giải pháp cho P(x)
- Nếu P được rút gọn thành Q, P *không khó hơn để giải quyết* so với Q
- Ví dụ đơn giản
  - Giải phương trình ax + b = 0 có thể được chuyển thành giải phương trình $ax^2 + bx + c = 0$

451

Rút gọn bài toán

- Ví dụ
  - Bài toán P
    - Cho tập các giá trị lô-gíc (đúng/sai), có tồn tại ít nhất một giá trị đúng ?
  - Bài toán Q
    - Cho tập các số nguyên, tổng của chúng có > 0 ?
  - Chuyển P thành Q
    - $(x_1,x_2,\ldots,x_n) = (y_1,y_2,\ldots,y_n)$ trong đó $y_i = 1$ nếu x_i = đúng, $y_i = 0$ nếu x_i = sai

452

Rút gọn bài toán

- Định nghĩa
  - Bài toán quyết định P có thể được *rút gọn thời gian đa thức* (*polynomial-time reducible*) thành bài toán quyết định Q, nếu tồn tại hàm f *tính được bởi thời gian đa thức* từ miền dữ liệu vào của P vào miền dữ liệu vào của Q sao cho:

    mọi x thuộc miền dữ liệu vào của P, P(x) đúng nếu và chỉ nếu Q(f(x)) đúng
  - P được rút gọn thời gian đa thức thành Q được ký hiệu $P \leq_p Q$

- Hàm f là một sự rút gọn P thành Q

- Nếu tồn tại thuật toán có độ phức tạp hàm đa thức cho Q thì cũng sẽ tồn tại thuật toán có độ phức tạp hàm đa thức cho P

453

NP-khó và NP-đầy đủ

- NP-khó (*NP-hard*)
  - Bài toán Q được gọi là *NP-khó* nếu thoả mãn:
    1. $\forall P \in NP, P \leq_p Q$

- NP-đầy đủ (*NP-complete*)
  - Bài toán Q được gọi là *NP-đầy đủ* nếu thoả mãn:
    1. $Q \in NP$
    2. $\forall P \in NP, P \leq_p Q$

- Nếu $P \leq_p Q$ và P là NP-đầy đủ thì Q cũng là NP-đầy đủ

454

Ví dụ

- Các bài toán NP-đầy đủ
  - *Tô màu đồ thị*: có thể tô màu đồ thị với nhiều nhất 3 màu sao cho hai đỉnh kề nhau bất kỳ có màu khác nhau ?
  - *Chu trình Hamilton*: cho đồ thị có tồn tại chu trình chứa tất cả các đỉnh của đồ thị ?
  - *Người du lịch*: một người du lịch muốn đi qua tất cả các thành phố với quảng đường ngắn nhất, với điều kiện mỗi thành phố chỉ đi qua một lần.
  - *SAT*: cho biểu thức lô-gíc gồm các biến lô-gíc $x_1, x_2, \ldots, x_n$ và các phép toán (AND, OR, NOT, $\Rightarrow$, $\Leftrightarrow$) và các dấu ngoặc. Có tồn tại phép gán các biến $x_1, x_2, \ldots, x_n$ sao cho biểu thức có giá trị đúng ?
  - …

455

Chứng minh bài toán NP-đầy đủ

- Tại sao phải chứng minh một bài toán là NP-đầy đủ ?
  - Không cần tìm kiếm các thuật toán giải quyết hiệu quả (thời gian đa thức)
  - Cần tìm các *thuật toán xấp xỉ (approximative algorithms)*

456

Chứng minh bài toán NP-đầy đủ

- Các bước chứng minh P là bài toán NP-đầy đủ
  1. Chứng minh P ∈ NP
  2. Chọn một bài toán Q là NP-đầy đủ (đã biết)
  3. Mô tả thuật toán tính hàm f chuyển các bài toán cụ thể của Q thành bài toán cụ thể của P
  4. Chứng minh rằng hàm f thoả mãn: Q(x) đúng nếu và chỉ nếu P(f(x)) đúng
  5. Chứng minh thuật toán tính hàm f có độ phức tạp hàm đa thức

- Cần ít nhất bài toán NP-đầy đủ đã biết
  - Định lý Cook
    - chứng minh bài toán đầu tiên là NP-đầy đủ

457

Định lý Cook

- Định lý Cook (1971)
  - SAT là bài toán NP-đầy đủ

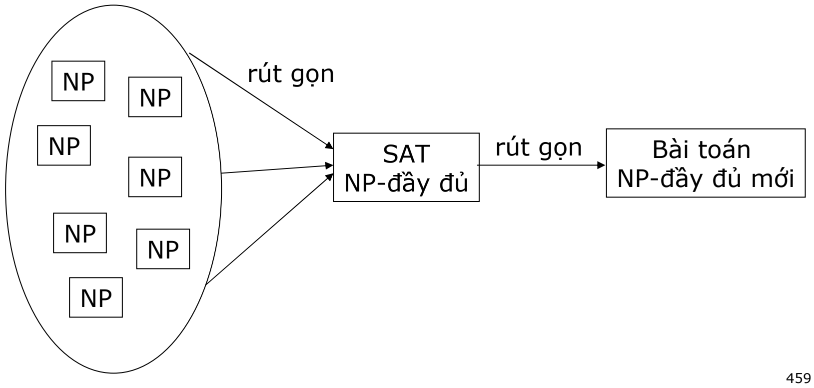
- Cho biểu thức lô-gíc gồm các biến lô-gíc x_1, x_2, …,x_n. Có tồn tại phép gán các biến x_1, x_2, …,x_n sao cho biểu thức có giá trị đúng ?

- Chứng minh
  - Dựa trên định nghĩa NP-đầy đủ
  - Không dựa vào phép rút gọn

458

Chứng minh bài toán NP-đầy đủ

- Sử dụng bài toán SAT để chứng minh các bài toán khác NP-đầy đủ



459

NP-đầy đủ

- Chi tiết hơn
  - Chapter 34: NP-Completeness, Introduction to algorithms, T.H. Cormen, C.E. Leiserson, R.R. Rivest, Mit Press 1990.

460

Thuật toán xấp xỉ (11)

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

Thuật toán xấp xỉ (approximation algorithms)

- Giải quyết các bài toán NP-đầy đủ
  - Thuật toán hàm mũ
    - Thuật toán quay lui
    - Không hiệu quả
  - Thuật toán xấp xỉ
    - Cho kết quả gần đúng
    - Độ phức tạp hàm đa thức
  - Thuật toán cho kết quả gần với kết quả tối ưu được gọi là *thuật toán xấp xỉ*

Thuật toán xấp xĩ

- Tỷ lệ xấp xĩ (approximation ratio)
  - Cần giải quyết bài toán tối ưu
    - Bài toán có thể có nhiều giải pháp
    - Cần tối ưu hàm một *hàm mục tiêu*
    - *Giải pháp tối ưu* là giải pháp mà cực đại (hoặc cự tiểu) hoá hàm mục tiêu
  - Đặt
    - C là giá trị của hàm mục tiêu của giải pháp xấp xĩ
    - C^* là giá trị của hàm mục tiêu của giải pháp tối ưu
  - Một thuật toán xấp xĩ có tỷ lệ xấp xĩ, ký hiệu $\rho(n)$, với bất kỳ dữ liệu vào kích thước n là:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

463

Thuật toán xấp xĩ

- Tỷ lệ xấp xĩ (2)
  - Định nghĩa tỷ lệ xấp xĩ đúng cho cả bài toán cực tiểu hoá và bài toán cực đại hoá
    - Bài toán cực tiểu hoá: $C \geq C^* > 0$, khi đó $\rho(n) = C/C^*$
    - Bài toán cực đại hoá: $C^* \geq C > 0$, khi đó $\rho(n) = C^*/C$
  - Một thuật toán với tỷ lệ xấp xĩ $\rho(n)$ được gọi là thuật toán xấp xĩ-$\rho(n)$
  - Luôn có: $\rho(n) \geq 1$
    - Nếu $\rho(n)$ càng nhỏ thì giải pháp xấp xĩ càng gần với giải pháp tối ưu
  - Thuật toán xấp xĩ-1 cho giải pháp tối ưu (nghĩa là $\rho(n) = 1$)

464

Một số thuật toán xấp xĩ

- Bài toán phủ các đỉnh (vertex cover problem)
- Bài toán người du lịch (traveling salesman problem)

465

Bài toán phủ các đỉnh

- Bài toán
  - Cho đồ thị vô hướng G = (V,E). *Phủ các đỉnh* của G là một tập con V' ⊂ V sao cho nếu (u,v) là một cạnh của G thì hoặc u ∈ V' hoặc v ∈ V'
  - Cần xác định phủ các đỉnh có kích thước nhỏ nhất
- Bài toán quyết định phủ các đỉnh tương ứng là NP-đầy đủ
  - Với k cho trước, có tồn tại phủ các đỉnh V' của đồ thị G sao cho |V'| ≤ k
- Rất phức tạp để tìm giải pháp tối ưu V'
- Tuy nhiên, thuật toán xấp xĩ rất đơn giản

466

Bài toán phủ các đỉnh

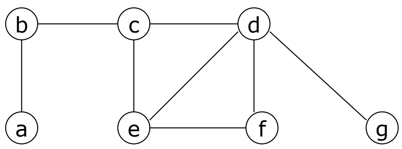
- Thuật toán xấp xỉ

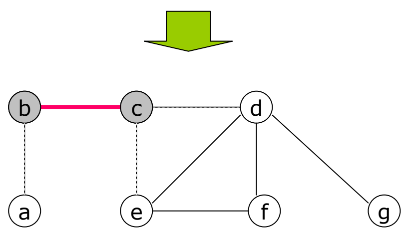
```
phucacdinh-xapxi (G=(V,E))
begin
    C = ∅
    U = E
    while (U ≠ ∅) do
        (u,v) là cạnh bất kỳ thuộc U
        C = C ∪ (u,v)
        xoá trong U tất cả các cạnh nối đến u hoặc v
    endwhile
    return (C)
end
```

  - Thuật toán có độ phức tạp là hàm tuyến tính

Bài toán phủ các đỉnh

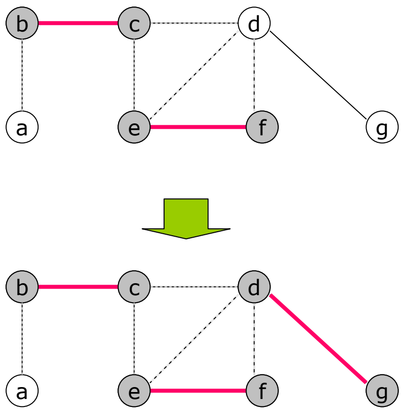
- Minh hoạ (1)





Bài toán phủ các đỉnh

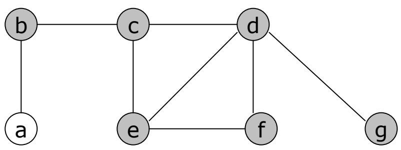
- Minh hoạ (2)



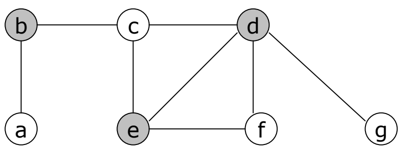
469

Bài toán phủ các đỉnh

- Minh hoạ (3)



Giải pháp xấp xĩ: b, c, d, e, f, g



Giải pháp tối ưu: b, d, e

470

Bài toán phủ các đỉnh

- Định lý: *phucacdinh-xapxi* là thuật toán xấp xỉ-2 ($\rho(n) \leq 2$)
- Chứng minh
  - Gọi A là tập các cạnh (u,v) được chọn bởi thuật toán. Bởi cách xây dựng A, thì không thể có hai cạnh bất kỳ của A có chung đỉnh (do bước xoá các cạnh nối đến u và v).
  - Vậy, mỗi bước lặp sẽ thêm vào C hai đỉnh u và v mới và |C| = 2 x |A| (*).
  - Gọi C^* là phủ các đỉnh với kích thước nhỏ nhất.
  - Vì hai cạnh của A không có đỉnh chung, nên một đỉnh của C^* chỉ nối đến nhiều nhất một cạnh của A.
  - Theo định nghĩa *phủ các đỉnh*, thì C^* phải chứa ít nhất một đỉnh của mỗi cạnh thuộc A. Vậy $|C^*| \geq |A|$ (**).
  - Kết hợp (*) và (**), ta có $|C| \leq 2|C^*|$

471

Bài toán người du lịch

- Bài toán
  - Cho đồ thị vô hướng liên thông hoàn toàn G=(V,E), nghĩa là giữa hai đỉnh bất kỳ luôn tồn tại cạnh nối chúng. Mỗi cạnh (u,v) có một trọng số $c(u,v) \geq 0$. Tìm chu trình Hamilton có tổng trọng số nhỏ nhất.
- Trong thực tế, đi từ đỉnh u đến trực tiếp một đỉnh v sẽ nhanh hơn đi từ u đến v qua một đỉnh trung gian w
- Tình huống này được mô tả bởi *bất đẳng thức tam giác* đối với hàm trọng số c
  - $c(u,v) \leq c(u,w) + c(w,v)$, với u, v, w là các đỉnh bất kỳ thuộc V
- Giả sử hàm trọng số c của bài toán thoả mãn bất đẳng thức tam giác
- Hầu hết các bài toán thực tế cũng thoả mãn bất đẳng thức tam giác
- Bài toán người du lịch thuộc lớp NP-đầy đủ

472

Bài toán người du lịch

- Thuật toán xấp xỉ

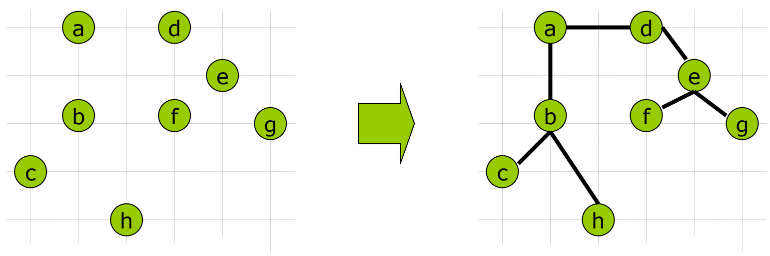
```
nguoidulich-xapxi (G)
begin
    Chọn một đỉnh r bất kỳ của G là đỉnh « gốc »
    Xác định cây khung nhỏ nhất T của G bởi thuật toán Prim
    Gọi L là danh sách các đỉnh khi duyệt T theo thứ tứ trước
    return (trả về chu trình Hamilton H khi
                thăm các đỉnh của L theo thứ tự)
end
```

  - Cây khung nhỏ nhất: chứa tất cả các đỉnh của đồ thị có tổng trọng số các cạnh nhỏ nhất
  - Duyệt thứ tự trước: một nút được thăm trước khi thăm các nút con của nó
  - Độ phức tạp thuật toán: $O(|V|^2)$
    - Độ phức tạp thuật toán Prim

473

Bài toán người du lịch

- Minh hoạ (1)



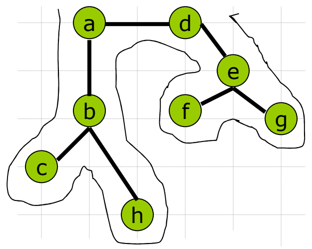
Đồ thị G với khoảng cách giữa các đỉnh là trọng số của cạnh tương ứng

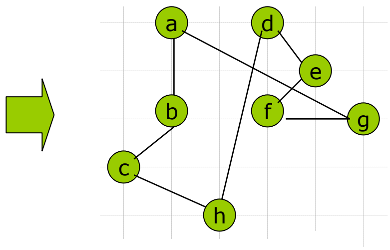
Cây khung nhỏ nhất, gốc là a

474

Bài toán người du lịch

- Minh hoạ (2)



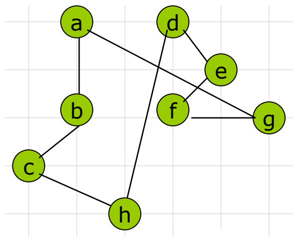


Duyệt cây T theo thứ tự trước:
a, b, c, b, h, b, a, d, e, f, e, g, e, d, a

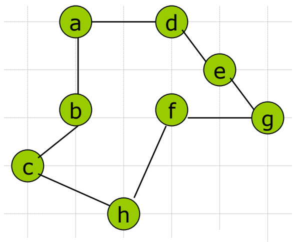
Chu trình Hamilton, các đỉnh của L: a, b, c, h, d, e, f, g, a

Bài toán người du lịch

- Minh hoạ (3)



Giải pháp xấp xĩ



Giải pháp tối ưu

Tổng trọng số của giải pháp xấp xĩ
≈ 123% của tổng trọng số của giải pháp tối ưu

Bài toán người du lịch

- Định lý: *nguoidulich-xapxi* là thuật toán xấp xỉ-2 ($\rho(n) \leq 2$)
- Chứng minh
  - H là chu trình Hamilton xác định bởi thuật toán xấp xỉ. H* là chu trình Hamilton có kích thước nhỏ nhất. Cần chứng minh $c(H) \leq 2c(H^*)$.
  - Bằng cách xoá một cạnh bất kỳ trong chu trình Hamilton H* ta có một cây khung T'. Vậy $c(T') \leq c(H^*)$.
  - Theo thuật toán T là cây khung nhỏ nhất. Nên $c(T) \leq c(T') \leq c(H^*)$.
  - Một phép *duyệt đầy đủ* cây T liệt kê tất cả các đỉnh khi gặp lần đầu và cả khi gặp lại sau khi đã duyệt cây con của chúng. Gọi W là phép duyệt đầy đủ T.
  - Chẳng hạn, trong ví dụ trước, phép duyệt đầy đủ liệt kê các đỉnh theo thứ tự trước:

    a, b, c, b, h, b, a, d, e, f, e, g, e, d, a

Bài toán người du lịch

- Chứng minh (tiếp)
  - Như thế, phép duyệt đầy đủ W của cây T sẽ thăm mỗi cạnh đúng hai lần. Vậy $c(W) = 2c(T) \leq 2c(H^*)$.
  - Ngoài ra, nhờ vào tính chất bất đẳng thức tam giác, chúng ta có thể xoá lần thăm một đỉnh của W mà không làm tăng trọng số của W:
    - nếu một đỉnh w được xoá giữa lần thăm đỉnh u và đỉnh v, thì danh sách mới các đỉnh sẽ đi trực tiếp từ u đến v (với trọng số $c(u,v) \leq c(u,w) + c(w,v)$).
  - Áp dụng xoá tất cả các lần thăm mỗi đỉnh của W, trừ lần thăm đầu tiên (nghĩa là xoá tất cả các lần thăm sau lần đầu tiên) và trừ lần thăm cuối cùng của đỉnh « gốc ».
  - Chẳng hạn, trong ví dụ trên, chúng la có danh sách:

    a, b, c, h, d, e, f, g, a
  - Danh sách này chính là chu trình Hamilton H cho bởi thuật toán xấp xỉ. Vậy chu trình Hamilton H có được bởi việc xoá các đỉnh của W (áp dụng tính chất bất đẳng thức tam giác).
  - Như thế: $c(H) \leq c(W) \leq 2c(H^*)$