ĐỀ CƯƠNG BÀI GIẢNG

BÀI 2: MỘT SỐ CHIẾN LƯỢC THIẾT KẾ GIẢI THUẬT

2.1 Giải thuật đệ quy

2.1.1. Khái niệm về đệ quy

Ta nói một đối tượng là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó.

Ví dụ: Trong toán học ta gặp các định nghĩa đệ quy sau:

+ Số tự nhiên:

- 1 là số tự nhiên.

- n là số tự nhiên nếu n-1 là số tự nhiên.

+ Hàm n giai thừa: n!

- 0! = 1

- Nếu n>0 thì n! = n(n-1)!

2.1.2. Giải thuật đệ quy và hàm đệ quy

Giải thuật đệ quy

Nếu lời giải của của một bài toán T được giải bằng lời giải của một bài toán T1, có dạng giống như T, thì lời giải đó được gọi là lời giải đệ quy. Giải thuật tương ứng với lời giải đệ quy gọi là giải thuật đệ quy.

Ở đây T1 có dạng giống T nhưng theo một nghĩa nào đó T1 phải "nhỏ" hơn T.

Chẳng hạn với bài toán tính n!, thì tính n! là bài toán T còn tính (n-1)! là bài toán T1 ta thấy T1 cùng dạng với T nhưng nhỏ hơn (n-1 < n).

Hay với bài toán tìm một từ trong quyển từ điển. Có thể nêu giải thuật như sau:

```
if (từ điển là một trang)
                tìm từ trong trang này
else
{               Mở từ điển vào trang "giữa"
                Xác định xem nửa nào của từ điển chứa từ cần tìm;
                if (từ đó nằm ở nửa trước)      tìm từ đó ở nửa trước
                else    tìm từ đó ở nửa sau.
```

}

Giải thuật này được gọi là giải thuật đệ quy. Việc tìm từ trong quyển từ điển được được giải quyết bằng bài toán nhỏ hơn đó là việc tìm từ trong một nửa thích hợp của quyển từ điển.

Ta thấy có hai điểm chính cần lưu ý:

1. Sau mỗi lần từ điển được tách làm đôi thì một nửa thích hợp sẽ lại được tìm bằng một chiến thuật như đã dùng trước đó (nửa này lại được tách đôi).

2. Có một trường hợp đặc biệt, đó là sau nhiều lần tách đôi từ điển chỉ còn một trang. Khi đó việc tách đôi ngừng lại và bài toán trở thành đủ nhỏ để ta có thể tìm từ mong muốn bằng cách tìm tuần tự. Trường hợp này gọi là trường hợp suy biến.

Hàm đệ quy

Với giải thuật tìm kiếm như trên ta viết một hàm tương ứng như sau:

```
SEARCH(dict, word)      //Tìm từ word trong từ điển dict
{
    if  (Từ điển chỉ còn là một trang)
            tìm từ word trong trang này
    else
    {       mở từ điển vào trang giữa
            xác định xem nửa nào của từ điển chứa từ word
            if (từ word nằm ở nửa trước của từ điển)
                    return  SEARCH(dict\{nửa sau}, word);
            else    return  SEARCH(dict\{nửa trước}, word);
    }
}
```

Hàm trên được gọi là thủ thục đệ quy. Nó có những đặc điểm cơ bản sau:

1. Trong thủ tục đệ quy có lời gọi đến chính thủ tục đó. ở đây trong thủ tục SEARCH có lời gọi call SEARCH (lời gọi này được gọi là lời gọi đệ quy).

2. Sau mỗi lần có lời gọi đệ quy thì kích thước của bài toán được thu nhỏ hơn trước. Ở đây khi có lời gọi call SEARCH thì kích từ điển chỉ còn bằng một nửa so với trước đó.

3. Có một trường hợp đặc biệt, trường hợp suy biến là khi lời gọi call SEARCH với từ điển dict chỉ còn là một trang. Khi trường hợp này xảy ra thì bài toán còn lại sẽ được giải quyết theo một cách khác hẳn (tìm từ word trong trang đó bằng cách tìm kiếm tuần tự) và việc gọi đệ quy cũng kết thúc. Chính tình trạng kích thước bài toán giảm dần sau mỗi lần gọi đệ quy, dẫn tới trường hợp suy biến.

Một số ngôn ngữ cấp cao như: Pascal, C, Algol v.v... cho phép viết các thủ tục đệ quy. Nếu thủ tục đệ quy chứa lời gọi đến chính nó thì gọi là đệ quy trực tiếp. Cũng có trường hợp thủ tục chứa lời gọi đến thủ tục khác mà ở thủ tục này lại chứa lời gọi đến nó. Trường hợp này gọi là đệ quy gián tiếp.

Thiết kế hàm đệ quy

Khi bài toán đang xét hoặc dữ liệu đang xử lý được định nghĩa dưới dạng đệ quy thì việc thiết kế các giải thuật đệ quy tỏ ra rất thuận lợi. Hầu như nó phản ánh rất sát nội dung của định nghĩa đó.

Không có giải thuật đệ quy vạn năng cho tất cả các bài toán đệ quy, nghĩa là mỗi bài toán cần thiết kế một giải thuật đệ quy riêng.

Ta xét một bài toán tính n!:

Hàm này được định nghĩa như sau:

$$Factorial(n) = \begin{cases} 1 & \text{nếu } n = 0 \\ n * Factorial(n-1) & \text{nếu } n > 0 \end{cases}$$

Giải thuật đệ quy được viết dưới dạng hàm dưới đây:

```
Factorial (n)
{
        if  (n= =0)  return 1;
        else    return  n*Factorial(n-1);
}
```

Trong hàm trên lời gọi đến nó nằm ở câu lệnh gán sau else.

Mỗi lần gọi đệ quy đến Factorial, thì giá trị của n giảm đi 1. Ví du, Factorial(4) gọi đến Factorial(3), gọi đến Factorial(2), gọi đến Factorial(1), gọi đến Factorial(0) đây là trường hợp suy biến, nó được tính theo cách đặc biệt Factorial(0) = 1.

2.2 Chia để trị

Ý tưởng về chia để trị

Chia để trị là một phương pháp rất hiệu quả khi ta đi thiết kế các thuật toán có cho những bài toán cỡ lớn và phức tạp. Từ những bài toán đầu vào rất lớn ta chia ra

thành những phần nhỏ hơn và đi tìm lời giải cho các bài toán nhỏ riêng biệt này, rồi sau đó tổng hợp những nghiệm bài toán nhỏ thành nghiệm của bài toán toàn cục. Với tư tưởng như vậy, một bài toán ta có thể phải chia nhỏ nhiều lần, chia cho đến khi bài toán được chia nhỏ chỉ còn lại rất nhỏ và lời giải khi đó là hiển nhiên. Tóm lại, những yếu tố chính của phương pháp chia để trị bao gồm:

1. Chia bài toán: Chia bài toán thành những bài toán nhỏ
2. Trị bài toán nhỏ: Giải những bài toán nhỏ này.
3. Kết hợp các nghiệm: Kết hợp những nghiệm của bài toán nhỏ cho nghiệm của bài toán lớn.

Có rất nhiều bài toán tính toán có thể giải hiệu quả bằng phương pháp chia để trị. Phương pháp này rấy mạnh khi thiết kế thuật toán, thậm trí có người còn khuyên việc đầu tiên giải bài toán là hãy đặt ra câu hỏi: Có tồn tại phương pháp chia để trị cho bài toán này hay không?

Thuật toán thiết kế theo cách chia để trị điển hình là thuật toán hồi quy, vì phần "Trị các bài toán nhỏ" đòi hỏi cùng một kỹ thuật trên những bài toán nhỏ hơn. Phân tích thời gian tính toán của thuật toán hồi quy có phức tạp hơn, nhưng ta sẽ chỉ ra rằng công cụ toán học phép truy toán rất có ích cho việc phân tích thuật toán loại này theo một cách tự nhiên.

Giải thuật tổng quát cho chiến lược chia để trị

divideAndConquer(A, x){

//Tìm lời giải x của bài toán A

if (A đủ nhỏ)

Solve(A);

else{

Chia bài toán thành các bài toán con:

$A_1, A_2, ..., A_m$

for (i=1; i<=m; i++)

divideAndConquer(A_i,x_i);

Kết hợp lời giải x_i của các bài toán con A_i để nhận được lời giải của bài toán A.

}

}

Thiết kế giải thuật

Xét bài toán tìm giá trị lớn nhất: Cho một dãy gồm n phần tử. Hãy tìm giá trị lớn nhất của các phần tử trong dãy bằng chiến lược chia để trị.

- **Bài toán**
  - ✓ Đầu vào: Dãy x[0...n-1] gồm n phần tử x[0], x[1], ..., x[n-1]
  - ✓ Đầu ra: Giá trị lớn nhất – max.
- **Áp dụng chiến lược chia để trị để thiết kế giải thuật.**
  - ✓ Chia dãy x[0...n-1] thành các dãy con: x[0...k] và x[k+1...n-1].
  - ✓ Tìm max trên các dãy con là bài toán cùng dạng, nhưng cỡ nhỏ hơn.
  - ✓ Để tìm max trên các dãy con ta tiếp tục chia đôi chúng (gọi đệ quy).
  - ✓ Quá trình chia đôi dừng lại khi nhận được các dãy con chỉ có 1 hoặc 2 phần tử.
- **Giải thuật**

```
max(x[], left, right){
    if (left == right) //Dãy có 1 phần tử
        return x[left];
    else if (left == right – 1){//Dãy có 2 phần tử
        if (x[left] > x[right]) return x[left];
        else return x[right];
    }
    else{ //Dãy có hơn 2 phần tử, chia đôi dãy
        mid = (left + right) / 2;
        maxLeft = max(x, left, mid);
        maxRight = max(x, mid + 1, right);
        if (maxLeft > maxRight) return maxLeft;
        else return maxRight;
    }
}
```

- **Độ phức tạp thuật toán**

Gọi T(n) là số phép toán so sánh cần thực hiện. Khi đó ta có:

$$T(n) = \begin{cases} T(n/2) + T(n/2) + 2 \ ; \ n > 2 \\ 1 \ ; \ n = 2 \\ 0 \ ; \ n = 1 \end{cases}$$

Với n= 2^k, thì:

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

$$= \sum_{i=1}^{k} 2^i - 2^{k-1} = 2^{k+1} - 2^{k-1} - 2 = \frac{3n}{2} - 2 .$$

Vậy T(n) $\in$ O(n)

2.3 Tham lam

Ý tưởng về tham lam

Phương pháp tham lam là kỹ thuật thiết kế thường được dùng để giải các bài toán tối ưu. Phương pháp được tiến hành trong nhiều bước. Tại mỗi bước, theo một lựa chọn nào đó (xác định bằng một hàm chọn), sẽ tìm một lời giải tối ưu cho bài toán nhỏ tương ứng. Lời giải của bài toán được bổ xung dần từng bước từ lời giải của các bài toán con.

Lời giải được xây dựng như thế có chắc là lời giải tối ưu của bài toán?

Các lời giải tìm được bằng phương pháp tham lam thường chỉ là chấp nhận được theo điều kiện nào đó, chưa chắc là tối ưu.

Cho trước một tập A gồm n đối tượng, ta cần phải chọn một tập con S của A. Với một tập con S được chọn ra thỏa mãn các yêu cầu của bài toán, ta gọi là một nghiệm chấp nhận được. Một hàm mục tiêu gắn mỗi nghiệm chấp nhận được với một giá trị. Nghiệm tối ưu là nghiệm chấp nhận được mà tại đó hàm mục tiêu đạt giá trị nhỏ nhất (lớn nhất).

Đặc trưng tham lam của phương pháp thể hiện bởi: trong mỗi bước, việc xử lí sẽ tuân theo một sự chọn lựa trước, không kể đến tình trạng không tốt có thể xảy ra khi thực hiện lựa chọn lúc đầu.

Những thuật toán tham lam được tạo ra rất dễ dàng, tương ứng với việc thể hiện nó trong ngôn ngữ lập trình là không phức tạp, nhưng có nhược điểm là đôi khi chúng không đảm bảo nghiệm đúng bài toán ban đầu. Nhược điểm này không làm giảm sự có ích của loại thuật toán này, vì những thuật toán loại này tính toán rất nhanh và tìm được lời giải bài toán, nhưng lời giải này cũng gần tối ưu. Ví dụ trong rất nhiều những bài toán thực tế ta không nghiên cứu được tất cả các trường hợp, đành phải chấp nhận phương án chủ quan do ta đặt ra mà thôi. Thiết lập những bài

toán như vậy đôi khi chỉ có 5% là nghiệm tồi trong những nghiệm tối ưu, so với vô hạn cách tìm nghiệm tối ưu.

Xét ví dụ với bài toán người du lịch có thể được giải bằng phương pháp tham lam. Bài toán được phát biểu như sau:

- Một người du lịch muốn tham quan n thành phố T_1, T_2, ..., T_n.
- Xuất phát từ một thành phố, đi qua tất cả các thành phố còn lại, mỗi thành phố 1 lần và trở về thành phố xuất phát.
- Gọi c_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j.
- Tìm một hành trình thỏa yêu cầu của bài toán sao cho tổng chi phí là thấp nhất.

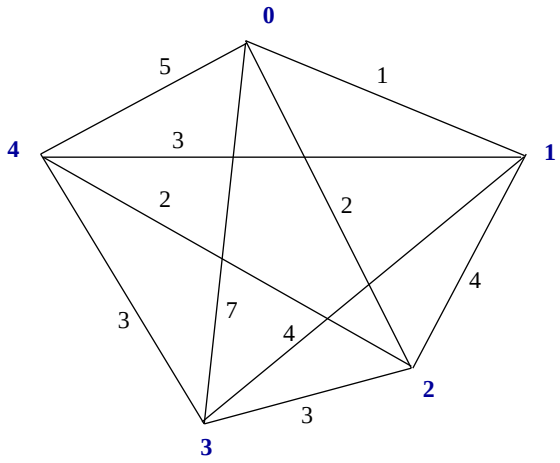
Giải thuật tổng quát cho phương pháp tham lam

```
void Greedy(S, G) //S là tập ứng viên, G tập nghiệm
{
    G = ∅;
    while (S != ∅)
    {
        n = bestGet(S); //chọn phần tử tốt nhất trong S
        S = S – {n};
        if (G ∪ {n} là chấp nhận được)
            G = G ∪ {n} ;
    }
}
```

Thiết kế giải thuật

Trong phần này chúng ta đi thiết kế giải thuật cho bài toán người du lịch đã nêu ở trên.

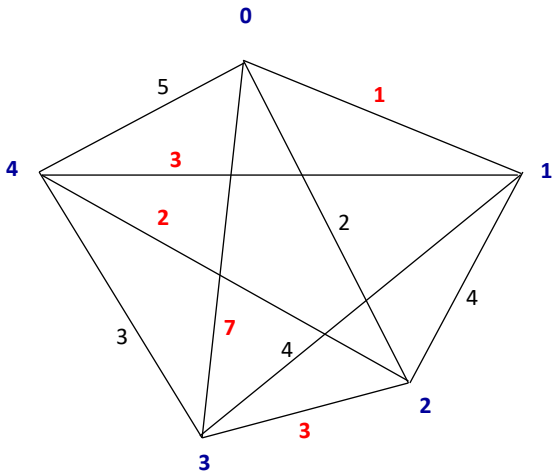
Bài toán người du lịch được đưa về bài toán tìm đường đi ngắn nhất trên đồ thị có trọng số.



Ma trận chi phí tương ứng với đồ thị này:

$$C = \begin{bmatrix} 0 & 1 & 2 & 7 & 5 \\ 1 & 0 & 4 & 4 & 3 \\ 2 & 4 & 0 & 3 & 2 \\ 7 & 4 & 3 & 0 & 3 \\ 5 & 3 & 2 & 3 & 0 \end{bmatrix}$$

Với chiến lược tại mỗi bước đi, ta sẽ chọn đi đến đỉnh lân cận sao cho chi phí đến đỉnh đó là thấp nhất.



- Phát biểu bài toán
    - Đầu vào:
        - n là số đỉnh của đồ thị.
        - u là đỉnh xuất phát.
        - Ma trận chi phí C = (c_{ij}).
    - Đầu ra:
        - tour là hành trình tốt nhất.
        - cost tổng chi phí ứng với tour tìm được.
- Mô tả:
    - v: đỉnh hiện tại đang được thăm.
    - B(v): tập các đỉnh kề với v và chưa được thăm.
    - (v, w): cạnh nối đỉnh v với đỉnh w có chi phí nhỏ nhất, w ∈ B(v).

Giải thuật giải quyết bài toán du lịch bao gồm các bước sau đây:

- Bước 1: *//Khởi tạo*

  tour = {}; cost = 0; v = u;
- Bước 2: *//Thăm tất cả các thành phố còn lại*

  for (k=1; k<n; k++){
- Bước 3: //Chọn cạnh kế tiếp

  Chọn (v,w);

  tour = tour ∪ {(v,w)};

  cost += C[v,w];

  Đỉnh w được gán nhãn đã thăm

  v = w *//Gán để xét bước kế tiếp*

  }
- Bước 4: //Hoàn thành

  tour = tour ∪ {(v,u)}

  cost += C[v,u];

2.4 Bài tập tự luyện

Bài 1: Cho mảng n phần tử đã được sắp xếp tăng dần và một phần tử x. Tìm x có trong mảng hay không? Nếu có x trong mảng thì cho kết quả là 1, ngược lại cho kết quả là 0. Hãy giải bài toán bằng thuật toán tìm kiếm nhị phân và cho biết ý tưởng này thuộc vào ý tưởng nào trong các ý tưởng: Chia để trị, đệ quy, tham lam. Hãy phân tích cụ thể hơn về các ý tưởng này cho cách giải bài toán. Hãy viết chương trình để hiện thực hóa ý tưởng đó và in các kết quả ra màn hình.

Bài 2: Cho một mảng a gồm n phần tử. Ta cần chuyển m phần tử đầu tiên của mảng với phần còn lại của mảng (n-m phần tử) mà không dùng một mảng phụ. Với m và n là các số bất kỳ nhập từ bàn phím và m<n. Hãy xây dựng ý tưởng để giải quyết bài toán này và cho biết nó thuộc vào các ý tưởng nào trong các ý tưởng: Chia để trị, đệ quy, tham lam. Hãy phân tích cụ thể hơn về các ý tưởng này cho cách giải bài toán. Hãy viết chương trình để hiện thực hóa ý tưởng đó và in các kết quả ra màn hình.

Bài 3: Viết chương trình nhập vào một số nguyên dương n. Hãy xây dựng giải thuật đệ quy và cho biết số n có bao nhiêu chữ số chẵn.

Bài 4:

- Việc tìm ước số chung lớn nhất của hai số nguyên dương p, q (p>q) được thực hiện như sau:
  - Tính số dư trong phép chia p cho q (r = p % q).
  - Nếu r = 0 thì ước số chung lớn nhất là q
  - Nếu r ≠ 0 thì gán cho p giá trị của q, gán cho q giá trị của r và lặp lại quá trình.

a) Xây dựng định nghĩa đệ qui cho việc tìm ước số chung lớn nhất của p, q nói trên.

b) Viết một giải thuật đệ qui và một giải thuật lặp thể hiện định nghĩa đó.

Bài 5: Cho n máy tính, chi phí nối máy i với máy j là c[i,j], ban đầu đã có sẵn một số cặp máy. Hãy tìm cách nối thêm dây cáp sao cho máy tính trong mạng là liên thông và chi phí nối mạng là nhỏ nhất. Hãy xây dựng ý tưởng để giải quyết bài toán này và cho biết nó thuộc vào các ý tưởng nào trong các ý tưởng sau đây hay không: Chia để trị, đệ quy, tham lam? Hãy viết chương trình để hiện thực hóa ý tưởng đó và in các kết quả ra màn hình.