BÀI 2

MỘT SỐ CHIẾN LƯỢC THIẾT KẾ GIẢI THUẬT

NỘI DUNG

1. Giải thuật đệ quy
2. Chia để trị
3. Tham lam
4. Bài tập

2.1. ĐỆ QUY

1. Giải thuật đệ quy và hàm đệ quy
2. Thiết kế giải thuật đệ quy

2.1.1. Giải thuật đệ quy và hàm đệ quy

2.1.1.1. Giải thuật đệ quy

- Nếu lời giải của của 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.
- Ví dụ tính "n giai thừa" - n! theo định nghĩa sau:
  - 0! = 1
  - Nếu n>0 thì n! = n(n-1)!
- Ở đây, 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).

Giải thuật đệ quy (tt)

- **Giải thuật đệ quy cho bài toán tìm từ trong từ điển**

```
search(dict, word){
   if (dict là một trang)
    Tìm từ word trong trang này
   else {
      Mở dict vào trang "giữa";
      Xác định nửa nào của dict chứa từ word;
      if (từ word nằm ở nửa trước)
         Tìm word ở nửa trước;
      else   Tìm từ word ở nửa sau;
    }
}
```

Giải thuật đệ quy (tt)

- **Nhận xét:**
  - 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).
  - 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**.

2.1.1.2. Hàm đệ quy

- Hàm đệ quy được cài đặt khi giải thuật được thiết kế là giải thuật đệ quy.

```
search(dict, word){
   if (Từ điển dict chỉ còn là một trang)
    Tìm từ word trong trang này
   else {
    Mở dict vào trang giữa
    Xác định nửa nào của dict chứa từ word
    if (từ word nằm ở nửa trước của dict)
        return search(dict\{nửa sau}, word);
    else
        return search(dict\{nửa trước}, word);
   }
}
```

Hàm đệ quy (tt)

- **Đặc điểm của hàm đệ quy:**
  - Trong hàm đệ quy có lời gọi đến chính nó. Trong hàm SEARCH có lệnh: return SEARCH(dict\{nửa trước}, word);
  - Sau mỗi lần có lời gọi đệ quy thì kích thước của bài toán được thu nhỏ hơn trước.
  - Trường hợp suy biến là khi lời gọi hàm SEARCH với từ điển dict chỉ còn là một trang. Trường hợp này 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.

2.1.2. Thiết kế giải thuật 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.
- Giải thuật đệ quy 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.

2.1.2.1. Tính n!

- Cách tính n! được định nghĩa như sau:
  - Nếu n = 0 -> n! = 1
  - Nếu n>0 -> n! = n*(n-1)!
- Giải thuật đệ quy được viết dưới dạng hàm

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

2.1.2.1. Tính n!

- **Nhận xét:**
  - Trong hàm **factorial(…)**, 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 dụ:
    - factorial(4) gọi đến factorial(3),
    - factorial(3) gọi đến factorial(2),
    - factorial(2) gọi đến factorial(1),
    - factorial(1) gọi đến factorial(0),
    - factorial(0), với n = 0, là trường hợp suy biến, theo định nghĩa factorial(0) = 1.

2.1.2.2. Bài toán dãy số fibonaci

- Dãy số fibonacci bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán được đặt ra như sau:
  - Các con thỏ không bao giờ chết.
  - Hai tháng sau khi được sinh ra một cặp thỏ mới sẽ sinh ra một cặp thỏ con.
  - Khi đã sinh sản, thì cứ sau mỗi tháng chúng lại sinh được một cặp con mới.
- Giả sử bắt đầu từ một cặp thỏ mới được sinh ra, hỏi đến tháng thứ n sẽ có bao nhiêu cặp?

Bài toán dãy số Fibonaci

- **Tính trực tiếp số cặp thỏ trong các tháng đầu tiên**

Chẳng hạn với n = 6, ta tính được

- **Tháng thứ 1**: **1 cặp** *(cặp ban đầu)*
- **Tháng thứ 2**: **1 cặp** *(cặp ban đầu vẫn chưa sinh con)*
- **Tháng thứ 3**: **2 cặp** *(đã có thêm 1 cặp con do cặp ban đầu đầu sinh ra)*
- **Tháng thứ 4**: **3 cặp** **(***cặp ban đầu vẫn sinh thêm***)**
- **Tháng thứ 5**: **5 cặp** **(***cặp con bắt đầu sinh***)**
- **Tháng thứ 6**: **8 cặp** **(***cặp con vẫn sinh tiếp***)**

Bài toán dãy số Fibonaci (tt)

- **Suy ra định nghĩa cách tính số cặp thỏ từ việc tính trực tiếp:**
  - **Đặt f(n) là số cặp thỏ ở tháng thứ n.**
  - **Ta thấy ở tháng thứ 1 (n = 1), và tháng thứ 2 (n = 2) luôn có 1 cặp**
  - **Chỉ những cặp thỏ đã có ở tháng thứ n - 2 mới sinh con ở tháng thứ n, nên số cặp thỏ ở tháng thứ n là:**

$$f(n) = f(n-2) + f(n-1).$$

Bài toán dãy số Fibonaci (tt)

- Vì vậy f(n) được định nghĩa dạng **ĐỆ QUI** như sau:

$$f(n) = \begin{cases} 1 \text{ nếu } n = 1 \text{ hoặc } n = 2 \\ f(n-2) + f(n-1) \text{ nếu } n > 2 \end{cases}$$

Dãy số f(n) ứng với các giá trị của n = 1, 2, 3, 4, 5, 6, 7, 8, ... có dạng: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... được gọi là dãy số fibonacci.

Bài toán dãy số Fibonaci (tt)

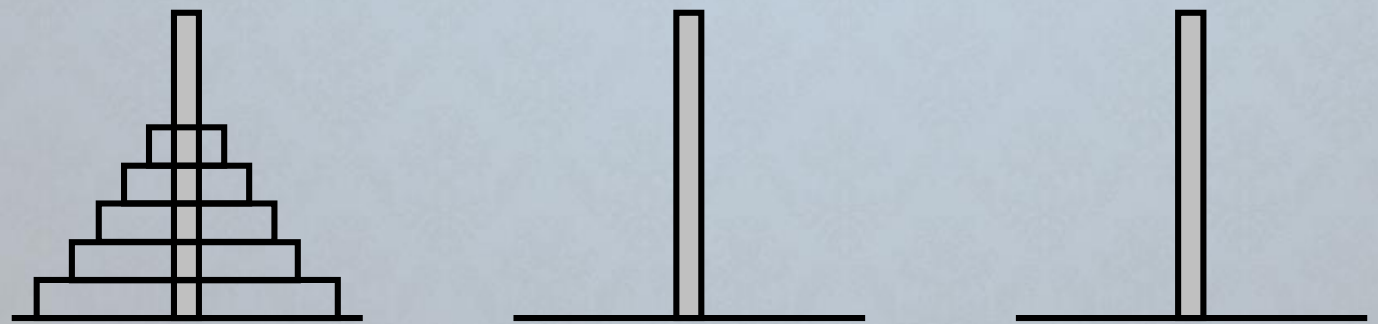
- Giải thuật đệ quy thể hiện việc tính f(n) được biểu diễn dưới dạng hàm.

```
long f(int n){
   if (n <= 2)
      return 1;
   else
      return f(n-2) + f(n-1);
}
```

- Ở đây trường hợp suy biến ứng với 2 giá trị f(1) = 1 và f(2) = 1.

2.1.2.3. Bài toán Tháp Hà Nội

- **Bài toán này mang tính chất là một trò chơi, nội dung như sau:**
  - **Có n đĩa, kích thước nhỏ dần, mỗi đĩa có lỗ ở giữa.**
  - **Có thể xếp chồng chúng lên nhau xuyên qua một cọc, đĩa to ở dưới, đĩa nhỏ ở trên để cuối cùng có một chồng đĩa**

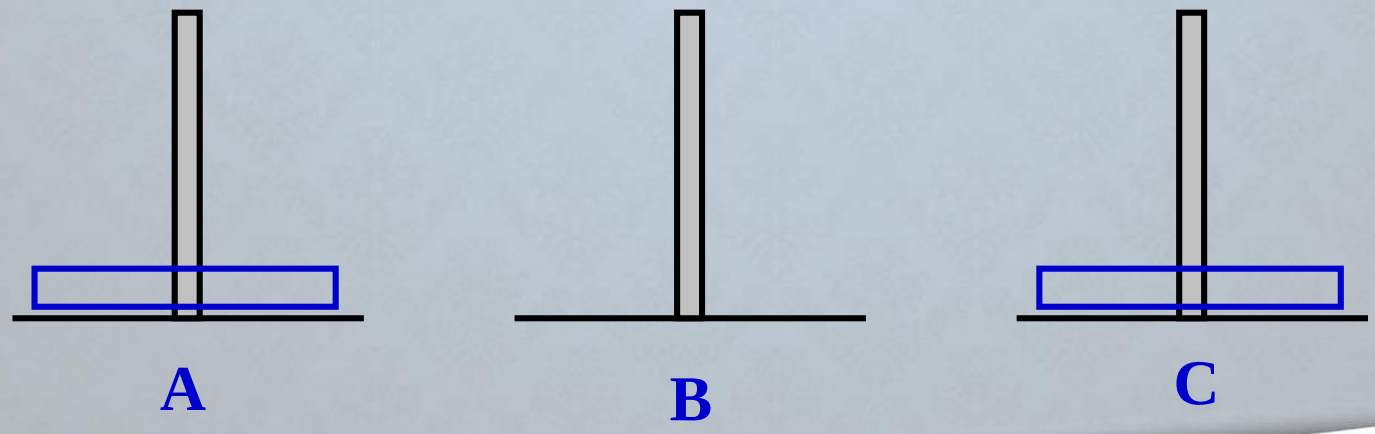


Bài toán Tháp Hà Nội (tt)

- Yêu cầu đặt ra là:
  - Chuyển chồng đĩa từ cọc A sang cọc khác, chẳng hạn cọc C, theo những điều kiện:
    - Mỗi lần chỉ được chuyển một đĩa.
    - Không khi nào có tình huống đĩa to ở trên đĩa nhỏ (dù là tạm thời).
    - Được phép sử dụng một cọc trung gian, chẳng hạn cọc B để đặt tạm đĩa.

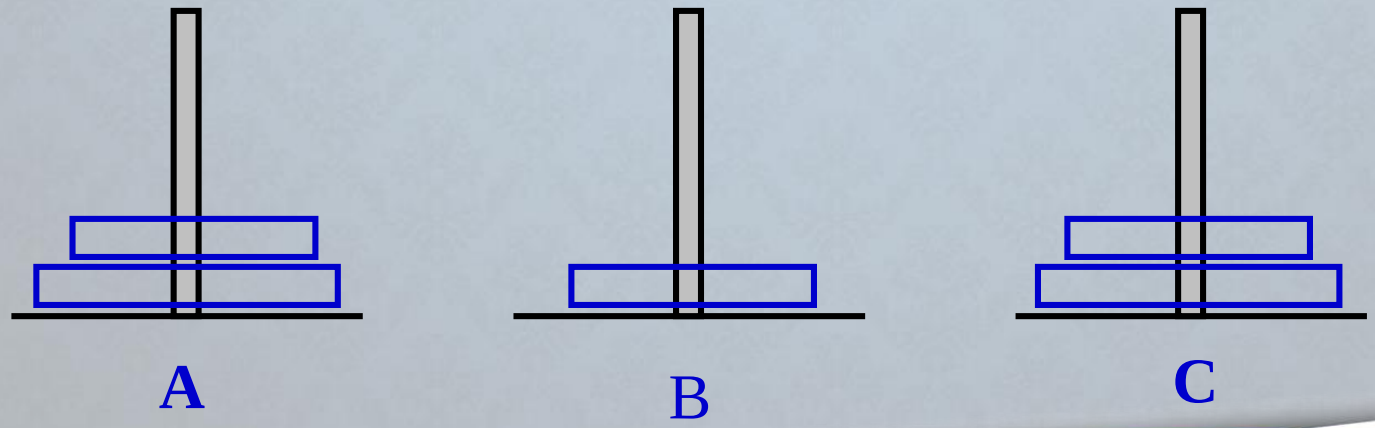
Bài toán Tháp Hà Nội (tt)

- Để đi tới cách giải tổng quát, trước hết ta giải quyết trực tiếp một số trường hợp đơn giản.
  - Trường hợp có 1 đĩa (n = 1):
    - Chuyển 1 đĩa từ cọc A sang cọc C.



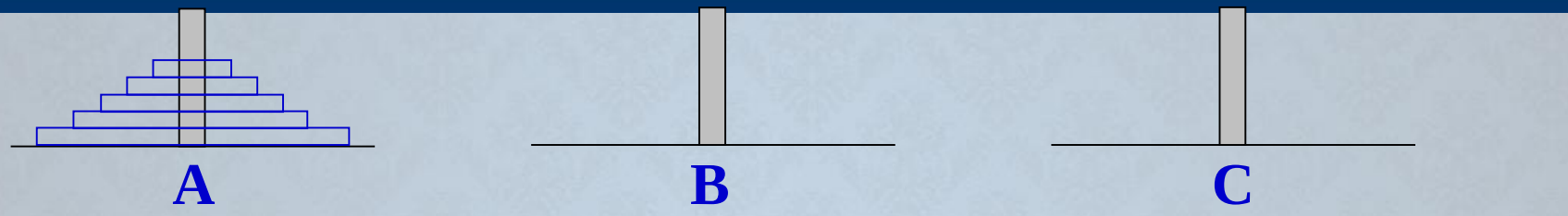
Bài toán Tháp Hà Nội (tt)

- **Trường hợp 2 đĩa:**
  - Chuyển 1 đĩa (đĩa thứ nhất) từ cọc A sang cọc B.
  - Chuyển 1 đĩa (đĩa thứ hai) từ cọc A sang cọc C.
  - Chuyển 1 đĩa (đĩa thứ nhất) từ cọc B sang cọc C.

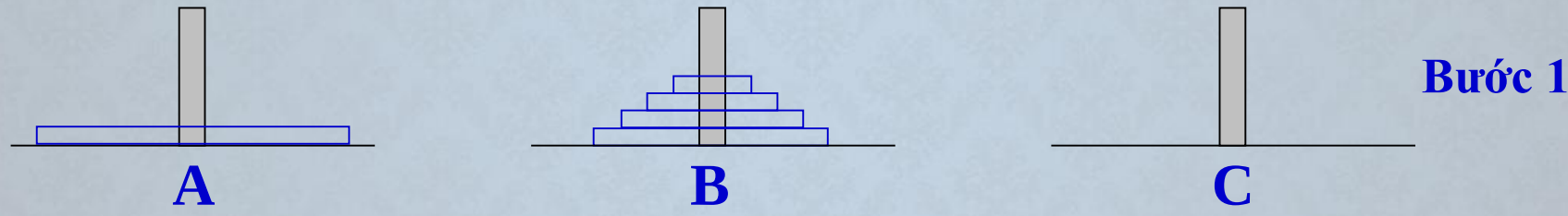


Bài toán Tháp Hà Nội (tt)

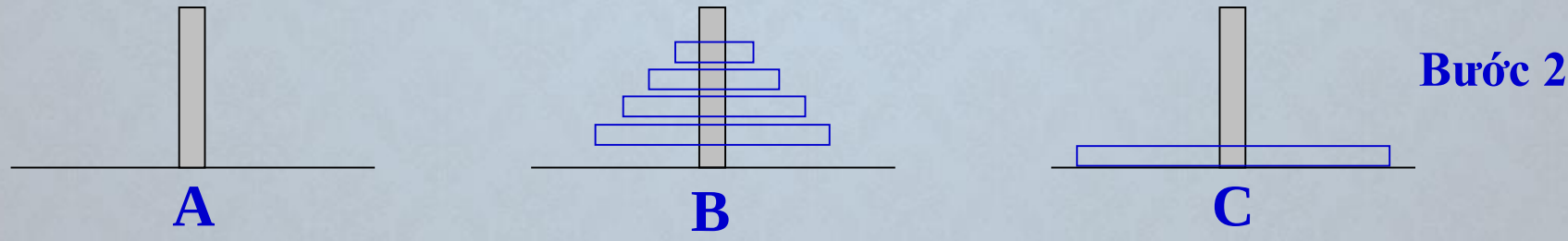
- Tổng quát với n đĩa (n > 2)
  - Ta thấy với trường hợp n đĩa (n > 2) nếu coi n - 1 đĩa ở trên, đóng vai trò như đĩa thứ nhất thì có thể xử lý giống như trường hợp 2 đĩa được, nghĩa là:
    - Chuyển n - 1 đĩa (phía trên) từ cọc A sang cọc B.
    - Chuyển 1 đĩa (dưới cùng) từ cọc A sang cọc C.
    - Chuyển n - 1 đĩa từ cọc B sang cọc C.



Bước 1



Bước 2



Bước 3

Bài toán Tháp Hà Nội (tt)

- **Nhận xét:**
  - Bài toán *"Tháp Hà Nội"* với n đĩa đã được dẫn đến bài toán tương tự với kích thước nhỏ hơn, chẳng hạn từ chỗ chuyển n đĩa từ cọc A sang cọc C nay là chuyển n - 1 đĩa từ cọc A sang cọc B và ở mức này thì giải thuật lại là:
    - Chuyển n-2 đĩa từ cọc A sang cọc C.
    - Chuyển 1 đĩa từ cọc A sang cọc B.
    - Chuyển n-2 đĩa từ cọc C sang cọc B.
  - và cứ như thế cho tới khi trường hợp suy biến xảy ra, đó là trường hợp ứng với bài toán chuyển 1 đĩa (n = 1).

Bài toán Tháp Hà Nội (tt)

- **Giải thuật đệ quy**

```
/*Chuyển n đĩa từ cọc A sang cọc C với B
là cọc trung gian*/
void Chuyen(n, A, B, C){
   if  (n == 1)
      Chuyển 1 đĩa từ A sang C;
   else{
      Chuyen(n-1, A, C, B);
      Chuyen(1, A, B, C);
      Chuyen(n-1, B, A, C);
   }
}
```

2.1.3. Hiệu lực của đệ quy

- Đệ quy là một kỹ thuật giải quyết bài toán khá hữu dụng.
- Việc thiết kế giải thuật cũng đơn giản vì nó khá giống với định nghĩa lời giải bài toán.
- Tuy nhiên:
  - Sử dụng đệ quy rất tốn bộ nhớ và thời gian
  - Nên sử dụng giải thuật lặp thay thế nếu được (khử đệ quy)
- Vẫn có những bài toán sử dụng đệ quy khá hữu ích: Giải thuật sắp xếp quick sort, các phép duyệt cây…

2.1.4. Bài tập

Bài 1:

- Xét định nghĩa đệ qui:

$$\text{Acker}(m,n) = \begin{cases} n+1 \text{ nếu } m = 0 \\ \text{Acker}(m-1,1) \text{ nếu } n = 0 \\ \text{Acker}(m-1, \text{Acker}(m, n-1)) \text{ còn lại} \end{cases}$$

- Hãy xác định Acker(1, 2)

- Viết hàm đệ qui thực hiện tính giá trị của hàm này.

Bài 2:

- Xét định nghĩa đệ qui:

$$F(x) = \begin{cases} \cos(x) \text{ nếu } x = 0 \\ x \text{ nếu } x < 0 \\ F(x-\pi) + F\left(x - \frac{\pi}{2}\right) \text{ còn lại} \end{cases}$$

- Hãy xác định F(3π/2)

- Viết hàm đệ qui thực hiện tính giá trị của hàm này.

Bài 3:

- 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 6:

- Viết một hàm lặp và một hàm đệ qui thực hiện việc in ngược một chuỗi ký tự. Ví dụ “PASCAL” thì in ra là “LACSAP”.

Bài 7:

- Xây dựng định nghĩa đệ qui cho việc đếm số chữ số của một số nguyên dương.
- Viết một hàm lặp và một hàm đệ qui thực hiện việc đếm số chữ số của một số nguyên dương.

2.2. chiến lược chia để trị - divide and conquer

1. Phương pháp chung
2. Thiết kế giải thuật
3. Ứng dụng

2.2.1. Phương pháp chung

- Nếu bài toán đủ nhỏ thì giải quyết bài toán (được kết quả trực tiếp).
- Ngược lại, chia bài toán thành các bài toán con cùng dạng (nhưng kích thước nhỏ hơn).
- Giải quyết các bài toán con (cũng bằng chiến lược chia để trị).
- Tổng hợp lời giải của các bài toán con để được lời giải của bài toán (ban đầu).
- Giải thuật được thiết kế theo chiến lược chia để trị sẽ là giải thuật đệ quy.

Phương pháp chung (tt)

- **Giải thuật tổng quát theo 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:
          A1, A2, ..., Am
      for (i=1; i<=m; i++)
         divideAndConquer(Ai,xi);
      Kết hợp lời giải xi của các bài toán con
      Ai để nhận được lời giải của bài toán A.
   }
}
```

Phương pháp chung (tt)

- "Chia" được hiểu là thực hiện các phép biến đổi, các phép toán để đưa việc giải quyết bài toán đã cho về việc giải quyết con cùng dạng, cỡ nhỏ hơn.
- "Trị" là giải quyết các bài toán con (cũng bằng chiến lược chia để trị -> gọi đệ quy).
- Một số giải thuật được thiết kế theo chiến lược chia để trị: Tháp Hà Nội, Tìm kiếm nhị phân, quick sort, merge sort *v.v...*

2.2.2. Thiết kế giải thuật

1. Giải thuật tìm giá trị lớn nhất
2. Giải thuật tính lũy thừa

2.2.2.1. Giải thuật tìm giá trị lớn nhất

- **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;
   }
}
```

2.2.2.2. Giải thuật tính lũy thừa

- **Bài toán**
  - **Đầu vào: Số nguyên a và số nguyên n**
  - **Đầu ra: a^n**
- **Áp dụng chiến lược chia để trị để thiết kế giải thuật.**
  - **Ta thấy: $a^n = a^{n/2} * a^{n/2}$ (n chẵn) = $a^{n/2} * a^{n/2} * a$ (n lẻ).**
  - **Tính $a^{n/2}$ và $a^{n/2}$ ta sẽ tính được a^n.**
  - **Tính $a^{n/2}$ là bài toán con cùng dạng, nhưng cỡ nhỏ hơn.**
  - **Quá trình dừng lại với việc tính $a^1 = a$.**

Giải thuật

```
power(a, n){
   if (n == 1)
      return a;
   else{
      x = power(a, n/2);
      if (n chẵn) return x * x;
      else return x * x * a;
   }
}
```

2.2.3. Bài tập

1. Cài đặt giải thuật tìm giá trị lớn nhất
2. Cài đặt giải thuật tính lũy thừa.

2.3. Chiến lược tham lam – greedy strategy

1. Phương pháp chung
2. Thiết kế giải thuật
3. Ứng dụng

2.3.1. Phương pháp chung

- Các bài toán tối ưu thường có một số lượng rất lớn nghiệm.
- Việc tìm ra nghiệm tối ưu tốn nhiều thời gian.
- Bài toán người du lịch:
  - 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.

Phương pháp chung (tt)

- **Chiến lược tham lam.**
  - **Lấy tiêu chuẩn toàn cục của bài toán làm tiêu chuẩn cục bộ.**
  - **Tại mỗi bước ta lựa chọn "nghiệm" tốt nhất trong số tất cả các "nghiệm" có thể có.**
- **Chiến lược tham lam thường không cho nghiệm tối ưu, nhưng sẽ cho nghiệm chấp nhận được.**

Phương pháp chung (tt)

- **Giải thuật tổng quát**

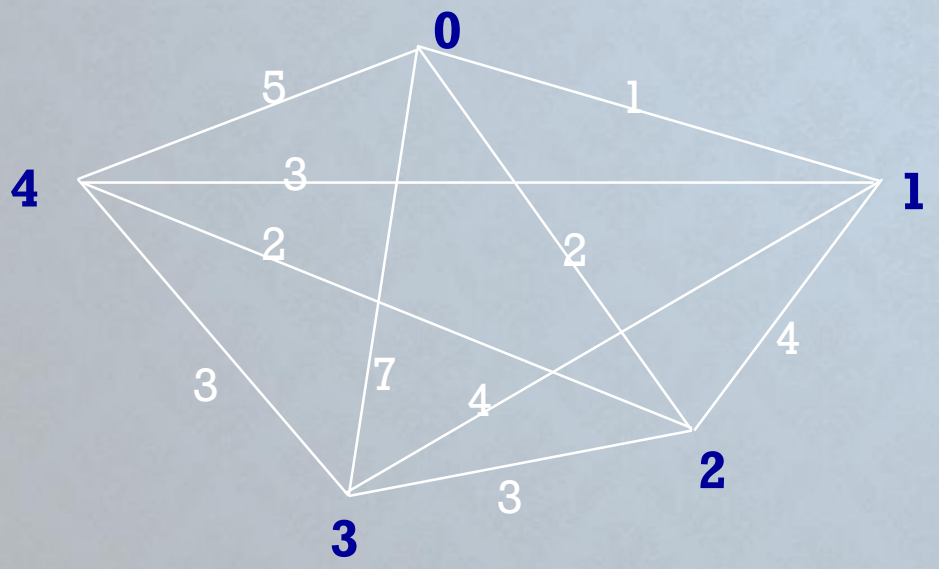
```
void Greedy(S, G) //S là tập ứng viên, G tập nghiệm
{
   G = ∅;
   while (S != ∅)
   {
    n = bestGet(S); //chọn phần tử tốt nhất trong S
    S = S - {n};
    if (G ∪ {n} là chấp nhận được)
        G = G ∪ {n} ;
   }
}
```

2.3.2. Thiết kế giải thuật

- **Bài toán người du lịch.**
- **Bài toán xếp ba lô.**

2.3.2.1. Bài toán người du lịch

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



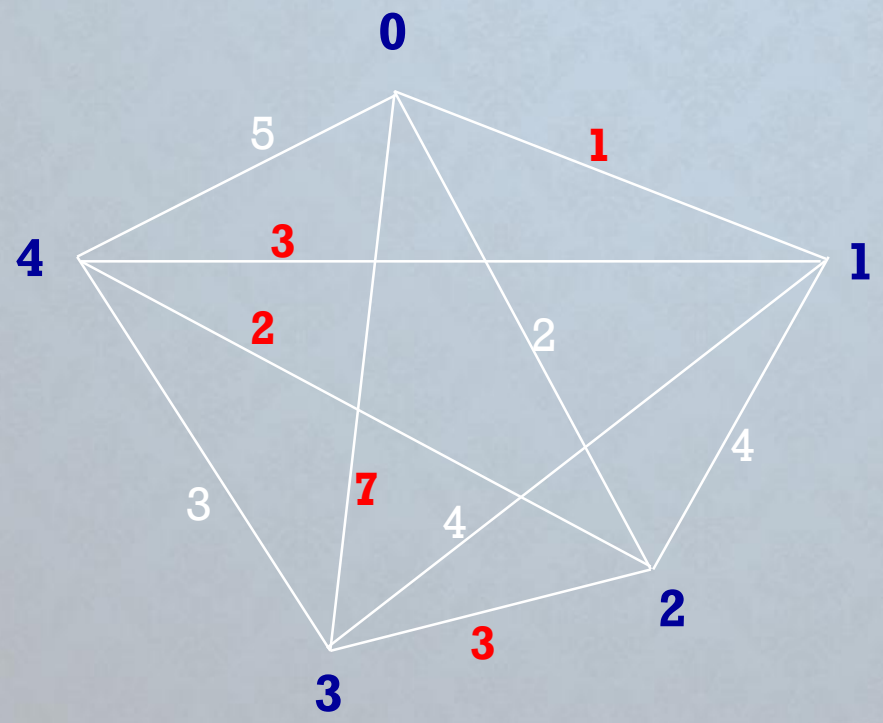
Ma trận chi phí

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

- Yêu cầu: Xuất phát từ 1 đỉnh, đi qua tất cả các đỉnh còn lại mỗi đỉnh 1 lần và trở về đỉnh xuất phát.
- Tìm đường đi với chi phí thấp nhất.

Bài toán người du lịch (tt)

- Tại mỗi bước đi, ta chọn đi đến đỉnh lân cận sao cho chi phí đến đỉnh đó là thấp nhất.



Bài toán người du lịch (tt)

- **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).**

Bài toán người du lịch (tt)

- **Giải thuật**
  - 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];
```

2.3.2.2. Bài toán xếp ba lô

- Một chiếc ba lô có thể chứa được các đồ vật với tổng khối lượng tối đa là w.
- Có n đồ vật ký hiệu v_1, ..., v_n, mỗi đồ vật v_i (i = 1, ..., n) có khối lượng là m_i và giá trị là c_i.
- Hãy xếp các đồ vật vào ba lô sao cho giá trị đạt được là lớn nhất.

Bài toán xếp ba lô (tt)

- **Ví dụ:**
  - **Ba lô chứa được tổng khối lượng là w = 20.**
  - **Có 5 đồ vật cho trong bảng dưới đây.**

Đồ vật	v1	v2	v3	v4	v5
Khối lượng	10	8	11	4	3
Giá trị	7	5	6	4	4

?

Thank you …!