# BÀI 2

# MỘT SỐ CHIẾN LƯỢC THIẾT KẾ GIẢI THUẬT




1



## NỘI DUNG



2



## MỘT SỐ KHÁI NIỆM

**Giải thuật là gì ?**

Giải thuật (hay còn gọi là thuật toán - tiếng Anh là Algorithms) là một tập hợp hữu hạn các chỉ thị để được thực thi theo một thứ tự nào đó để thu được kết quả mong muốn. Nói chung thì giải thuật là độc lập với các ngôn ngữ lập trình, tức là một giải thuật có thể được triển khai trong nhiều ngôn ngữ lập trình khác nhau.

3



## MỘT SỐ KHÁI NIỆM

Xuất phát từ quan điểm của cấu trúc dữ liệu, dưới đây là một số giải thuật quan trọng:

- **Giải thuật Tìm kiếm**: Giải thuật để tìm kiếm một phần tử trong một cấu trúc dữ liệu.
- **Giải thuật Sắp xếp**: Giải thuật để sắp xếp các phần tử theo thứ tự nào đó.
- **Giải thuật Chèn**: Giải thuật để chèn phần từ vào trong một cấu trúc dữ liệu.
- **Giải thuật Cập nhật**: Giải thuật để cập nhật (hay update) một phần tử đã tồn tại trong một cấu trúc dữ liệu.
- **Giải thuật Xóa**: Giải thuật để xóa một phần tử đang tồn tại từ một cấu trúc dữ liệu.

4



**Đặc điểm của giải thuật**

*Không phải tất cả các thủ tục có thể được gọi là một giải thuật. Một giải thuật nên có các đặc điểm sau:*

- **Tính xác định**: Giải thuật nên rõ ràng và không mơ hồ. Mỗi một giai đoạn (hay mỗi bước) nên rõ ràng và chỉ mang một mục đích nhất định.
- **Dữ liệu đầu vào xác định**: Một giải thuật nên có 0 hoặc nhiều hơn dữ liệu đầu vào đã xác định.
- **Kết quả đầu ra**: Một giải thuật nên có một hoặc nhiều dữ liệu đầu ra đã xác định, và nên kết nối với kiểu kết quả bạn mong muốn.
- **Tính dừng**: Các giải thuật phải kết thúc sau một số hữu hạn các bước.
- **Tính hiệu quả**: Một giải thuật nên là có thể thi hành được với các nguồn có sẵn, tức là có khả năng giải quyết hiệu quả vấn đề trong điều kiện thời gian và tài nguyên cho phép.
- **Tính phổ biến**: Một giải thuật có tính phổ biến nếu giải thuật này có thể giải quyết được một lớp các vấn đề tương tự.
- **Độc lập**: Một giải thuật nên có các chỉ thị độc lập với bất kỳ phần code lập trình nào.

5



## MỘT SỐ KHÁI NIỆM

**Cách viết một giải thuật** ?

Trong thực tế, không có bất kỳ tiêu chuẩn nào cho trước để viết các giải thuật. Như chúng ta đã biết, các ngôn ngữ lập trình đều có các vòng lặp (**do, for, while**) và các lệnh điều khiển luồng (**if-else**), … Chúng ta có thể sử dụng những lệnh này để viết một giải thuật.

Chúng ta viết các giải thuật theo cách thức là theo từng bước một. Viết giải thuật là một tiến trình và được thực thi sau khi bạn đã định vị rõ ràng vấn đề cần giải quyết.

Từ việc định vị vấn đề, chúng ta sẽ thiết kế ra giải pháp để giải quyết vấn đề đó và sau đó là viết giải thuật.

6

## MỘT SỐ KHÁI NIỆM

Ví dụ viết giải thuật

**Bài toán**: Thiết kế một giải thuật để cộng hai số và hiển thị kết quả.

```
Bước 1: Bắt đầu
Bước 2: Khai báo ba số a, b & c
Bước 3: Định nghĩa các giá trị của a & b
Bước 4: Cộng các giá trị của a & b
Bước 5: Lưu trữ kết quả của Bước 4 vào biến c
Bước 6: In biến c
Bước 7: Kết thúc
```



7



## MỘT SỐ KHÁI NIỆM

Ví dụ viết giải thuật

**Bài toán**: Thiết kế một giải thuật để cộng hai số và hiển thị kết quả.

Các giải thuật nói cho lập trình viên cách để viết code. Chúng ta cũng có thể viết một giải thuật cho bài toán trên như sau:

**Bước 1**: Bắt đầu

**Bước 2**: Lấy giá trị của **a** & **b**

**Bước 3**: c ← a + b

**Bước 4**: Hiển thị c

**Bước 5**: Kết thúc



8

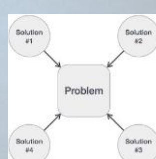


## MỘT SỐ KHÁI NIỆM

Chúng ta viết một giải thuật để tìm giải pháp xử lý một bài toán nào đó.

Một bài toán có thể được giải theo nhiều cách khác nhau

*Do đó, một bài toán có thể sẽ có nhiều lời giải. Vậy lời giải nào sẽ là thích hợp nhất cho bài toán đó*






9



## 2.1. ĐỆ QUY

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



10



## 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).



11



## 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;
   }
}
```



12

## 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 (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);
}
```







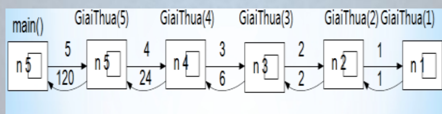


## 2.1.2.1. Tính n!

- 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.

### 2.1.2.1. Tính n!

Hoạt động của chương trình với n = 5







19



### 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?




20



### 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*)




21



### 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).




22



### 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, 9, 10,...
có dạng: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,...
được gọi là dãy số fibonacci.




23



### 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);
}
```

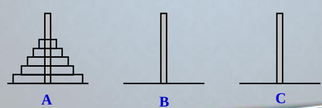
- Ở đây trường hợp suy biến ứng với 2 giá trị f(1) = 1 và f(2) = 1.




24

## 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








25



## 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:
    1. Mỗi lần chỉ được chuyển một đĩa.
    2. Không khi nào có tình huống đĩa to ở trên đĩa nhỏ (dù là tạm thời).
    3. Được phép sử dụng một cọc trung gian, chẳng hạn cọc B để đặt tạm đĩa.



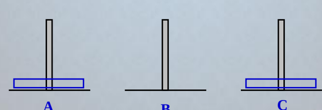


26



## 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.





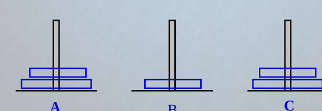



27



## 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.








28



## Bài toán Tháp Hà Nội (tt)

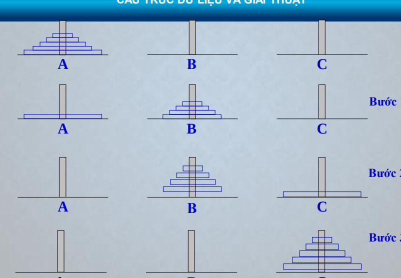
- Tổng quát với n đĩa (n > 2)
  - Ta thấy với trường hợp n đĩa (n > 2) nếu coi n - 1 đĩa ở trên, đóng vai trò như đĩa thứ nhất thì có thể xử lý giống như trường hợp 2 đĩa được, nghĩa là:
    - Chuyển n - 1 đĩa (phía trên) từ cọc A sang cọc B.
    - Chuyển 1 đĩa (dưới cùng) từ cọc A sang cọc C.
    -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





29










30

## 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:

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

- 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 4:**

- 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 5:**

- 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









**Phương phát chung**

Phương pháp chia để trị (Divide and Conquer) là một phương pháp quan trọng trong việc thiết kế các giải thuật. Ý tưởng của phương pháp này khá đơn giản và rất dễ hiểu:

Khi cần giải quyết một bài toán, ta sẽ tiến hành chia bài toán đó thành các bài toán con nhỏ hơn.

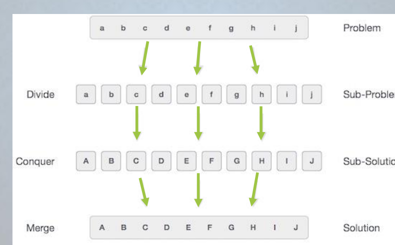
Tiếp tục chia cho đến khi các bài toán nhỏ này không thể chia thêm nữa, khi đó ta sẽ giải quyết các bài toán nhỏ nhất này và cuối cùng kết hợp giải pháp của tất cả các bài toán nhỏ để tìm ra giải pháp của bài toán ban đầu.




















**Phương phát chung (tt)**

Chúng ta có thể hiểu giải thuật chia để trị qua 3 tiến trình sau:

**Tiến trình 1: Chia nhỏ (Divide/Break)**

Trong bước này, chúng ta chia bài toán ban đầu thành các bài toán con. Mỗi bài toán con nên là một phần của bài toán ban đầu. Nói chung, bước này sử dụng phương pháp đệ qui để chia nhỏ các bài toán cho đến khi không thể chia thêm nữa. Khi đó, các bài toán con được gọi là "atomic – nguyên tử", nhưng chúng vẫn biểu diễn một phần nào đó của bài toán ban đầu.

**Tiến trình 2: Giải bài toán con (Conquer/Solve)**

Trong bước này, các bài toán con được giải.

**Tiến trình 3: Kết hợp lời giải (Merge/Combine)**

Sau khi các bài toán con đã được giải, trong bước này chúng ta sẽ kết hợp chúng một cách đệ qui để tìm ra giải pháp cho bài toán ban đầu.









**Phương phát chung (tt)**

Giải thuật chia để trị tồn tại hai vấn đề

1. Làm thế nào để chia tách bài toán một cách hợp lý thành các bài toán con, bởi vì nếu các bài toán con được giải quyết bằng các thuật toán khác nhau thì sẽ rất phức tạp.

2. Việc kết hợp lời giải các bài toán con được thực hiện như thế nào.

## 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...*



43



## 2.2.2. Thiết kế giải thuật

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



44



## 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.



45



## 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;
  }
}
```



46



## 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 $a^1 = a$.



47



## 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;
  }
}
```



48

## Giải thuật QUICKSORT

- **Bước 1(chia)**: Thuật toán quicksort chia danh sách cần sắp xếp mảng array[1..n] thành hai danh sách con có kích thước tương đối bằng nhau nhờ chỉ số của phần tử gọi là chốt, ta có thể chọn chốt là phần tử ở giữa, ở cuối, ở đầu hoặc phần tử ngẫu nhiên nào trong mảng.
  **Bước 2(trị)**: Sau khi đã chia thành 2 mảng dựa vào phần tử chốt nhiệm vụ của bước này là phải sắp xếp sao cho: những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách đứng sau(Trường hợp sắp xếp tăng dần). Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1
  **Bước 3**: Bằng việc lặp các bước giải quyết các bài toán con trên ta sẽ thu được kết quả là mảng sẽ được sắp xếp.

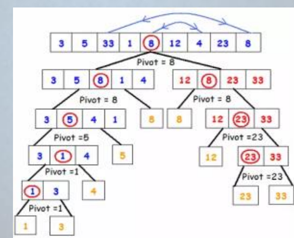


49



## Giải thuật QUICKSORT

Hình ảnh mô tả thuật toán trên dãy số đa cho





50



**Giải thuật QUICKSORT**

```
// hàm giải quyết việc sắp xếp các phần tử ở hai đầu của mảng
// dựa vào phần tử chốt là phần tử cuối mảng
    int partition(int arr[], int low, int high)
    {           // chốt được chọn ở đây là phần tử cuối mảng
        int pivot = arr[high];
        int i = (low-1);
        for (int j=low; j<high; j++)
        {  // nếu phần tử nhỏ hơn hoặc bằng với chốt
            if (arr[j] <= pivot)
            {   i++;
               // đổi chỗ arr[i] và arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // đổi chỗ arr[i+1] và arr[high] (chốt)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;
        // trả về chỉ số của chốt
        return i+1;
    }
```



51



**Giải thuật QUICKSORT**

```
  // Hàm thực hiện quicksort
void sort(int arr[], int low, int high)
{ // nếu chỉ số của đầu mảng nhỏ hơn chỉ số cuối mảng
    if (low < high)
    {   // tìm chỉ số của chốt sau khi đã thực hiện sắp xếp
        int pi = partition(arr, low, high);
      // lặp lại các bước với mảng từ phần tử đầu tiên đến chốt - 1
      // và từ chốt + 1 đến phần tử cuối cùng của mảng.
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```



52



## 2.2.3. Bài tập

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



53



## 2.3. Chiến lược tham lam – greedy strategy

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



54

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 2.3.1. Phương pháp chung

**Giải thuật tham lam là gì ?**

Tham lam là một trong những phương pháp phổ biến nhất để thiết kế giải thuật.

(*dựa trên câu truyện dân gian thì sẽ có câu chuyện như thế này: trên một mâm cỗ có nhiều món ăn, món nào ngon nhất ta sẽ ăn trước, ăn hết món đó ta sẽ chuyển sang món ngon thứ hai, và chuyển tiếp sang món thứ ba,...*)

Rất nhiều giải thuật nổi tiếng được thiết kế dựa trên ý tưởng tham lam, ví dụ như giải thuật cây khung nhỏ nhất của Dijkstra, giải thuật cây khung nhỏ nhất của Kruskal,



55

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 2.3.1. Phương pháp chung

- Giải thuật tham lam là giải thuật tối ưu hóa tổ hợp. Giải thuật tìm kiếm, lựa chọn giải pháp tối ưu địa phương ở mỗi bước với hi vọng tìm được giải pháp tối ưu toàn cục.
- Giải thuật tham lam lựa chọn giải pháp nào được cho là tốt nhất ở thời điểm hiện tại và sau đó giải bài toán con nảy sinh từ việc thực hiện lựa chọn đó.
- Lựa chọn của giải thuật tham lam có thể phụ thuộc vào lựa chọn trước đó.
- Việc quyết định sớm và thay đổi hướng đi của giải thuật cùng với việc không bao giờ xét lại các quyết định cũ sẽ dẫn đến kết quả là giải thuật này không tối ưu để tìm giải pháp toàn cục.



56

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 2.3.1. Phương pháp chung

**Ví dụ: Bài toán đếm số đồng tiền**

Yêu cầu là hãy lựa chọn số lượng đồng tiền nhỏ nhất có thể sao cho tổng mệnh giá của các đồng tiền này bằng với một lượng tiền cho trước.

Nếu tiền đồng có các mệnh giá lần lượt là 1, 2, 5, và 10 xu và lượng tiền cho trước là 18 xu thì giải thuật tham lam thực hiện như sau:

- **Bước 1**: Chọn đồng 10 xu, do đó sẽ còn 18 – 10 = 8 xu.
- **Bước 2**: Chọn đồng 5 xu, do đó sẽ còn là 3 xu.
- **Bước 3**: Chọn đồng 2 xu, còn lại là 1 xu.
- **Bước 4**: Cuối cùng chọn đồng 1 xu và giải xong bài toán.

Chúng ta thấy rằng cách làm trên là tốt, và số lượng đồng tiền cần phải lựa chọn là 4 đồng tiền



57

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 2.3.1. Phương pháp chung

**Ví dụ: Bài toán đếm số đồng tiền**

Tuy nhiên, nếu thay đổi bài toán 1 chút, vẫn giữ cách tiếp cận như trên dẫn đến lời giải không tốt

Chẳng hạn, một hệ thống tiền tệ khác có các đồng tiền có mệnh giá lần lượt là 1, 7 và 10 xu và lượng tiền cho trước là 15 xu.

Theo giải thuật tham lam thì số đồng tiền cần chọn sẽ nhiều hơn 4. Với giải thuật tham lam thì: 10 + 1 + 1 +1 + 1 + 1, vậy tổng cộng là 6 đồng tiền.

Trong khi cùng bài toán như trên có thể được xử lý bằng việc chỉ chọn 3 đồng tiền (7 + 7 +1).

*Do đó chúng ta có thể kết luận rằng, giải thuật tham lam tìm kiếm giải pháp tối ưu ở mỗi bước nhưng lại có thể thất bại trong việc tìm ra giải pháp tối ưu toàn cục.*



58

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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.**



59

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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.**



60

- 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
```



61

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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 = $\{a_1, a_2, ..., a_n\}$ 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: $\forall a_1 \in S, a_2 \in S$: $s(a_1) \le s(a_2) \Rightarrow f(a_1) \le f(s_2)$

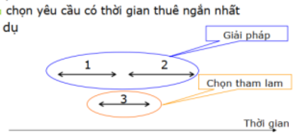


62

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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

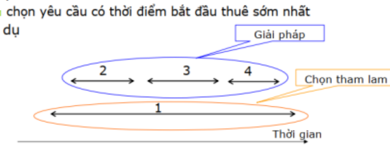


63

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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






64

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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



65

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## Thuê xe

- Chứng minh tính tối ưu của thuật toán
  - Giả sử F=$\{x_1, x_2, ..., x_p\}$ là giải pháp đạt được bởi thuật toán tham lam và G=$\{y_1, y_2, ..., y_q\}$ với $q \ge p$ 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 $x_k \ne y_k$. $x_k$ 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 $f(x_k) \le f(y_k)$. Thay G bởi G'=$\{y_1, y_2, ..., y_{k-1}, x_k, y_{k+1}, ..., y_q\}$ 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



66

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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
```





67

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## 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





68

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

### 2.3.2. Thiết kế giải thuật

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



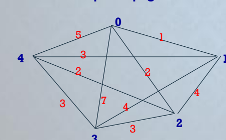


69

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

### 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.



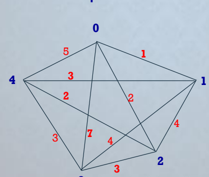


70

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

### 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.








71

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

### 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).





72

## 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 |

?





## 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
```





**01.04.22**

CÁC BẠN LẤY BÀI THỰC HÀNH lab.02 TRONG LỚP ĐIỆN TỬ VÀ THỰC HIỆN BÀI 1, 2, 3

## THUẬT TOÁN QUAY LUI (BACKTRACKING)

Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy. Người đầu tiên đề ra thuật ngữ này (backtrack) là nhà toán học người Mỹ D. H. Lehmer vào những năm 1950.

**Tư tưởng**

Dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.



79

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## THUẬT TOÁN QUAY LUI (BACKTRACKING)

Các bước trong việc liệt kê cấu hình dạng X[1...n]:

- Xét tất cả các giá trị X[1] có thể nhận, thử X[1] nhận các giá trị đó. Với mỗi giá trị của X[1] ta sẽ:
- Xét tất cả giá trị X[2] có thể nhận, lại thử X[2] cho các giá trị đó. Với mỗi giá trị X[2] lại xét khả năng giá trị của X[3]...tiếp tục như vậy cho tới bước:
- ...
- ....
- Xét tất cả giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt giá trị đó.
- Thông báo cấu hình tìm được.

*Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu (Depth-First Search).*



80

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

**Mô hình thuật toán -** Mã giả cho thuật toán quay lui

```
Backtracking(k)
{
    for([Mỗi phương án chọn i(thuộc tập D)])
    {   if ([Chấp nhận i])
        {   [Chọn i cho X[k]];
            if ([Thành công])
            {   [Đưa ra kết quả];
            } else
            {
                Backtracking(k+1);
                [Bỏ chọn i cho X[k]];
            }
        }
    }
}
```



81

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

**Ví dụ: Trò chơi Sudoku**

Sudoku là một trò chơi khá phổ biến và chắc ai cũng biết. Trò chơi như sau: có một hình vuông được chia thành 9x9 ô vuông con. Mỗi ô vuông con có giá trị trong khoảng từ 1 đến 9.

Ban đầu hình vuông có một số ô vuông con cho trước (có điền sẵn số) và còn lại là trống. Hãy điền các số từ 1-9 vào các ô con lại sao cho: hàng ngang là các số khác nhau từ 1 đến 9, hàng dọc là các số khác nhau từ 1 đến 9, và mỗi khối 3x3 chính là các số khác nhau từ 1 đến 9.

Áp dụng quay lui để giải bài toán sudoku.

Ý tưởng: Mỗi bước tìm tập các giá trị khả dĩ để điền vào ô trống, và sau đó đệ quy để điền ô tiếp theo. Giả mã của thuật toán (ở đây chú ý mảng chỉ có kích thước 9×9×9)



82

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

**Ví dụ: Trò chơi Sudoku**

```
void solveSudoku(int S[][9], int x, int y){
    if(y == 9){
        if(x == 8){
            printSolution(S);
            exit(0);
        } else {
            solveSudoku(S, x+1,0);
        }
    } else if(S[x][y] == 0){
        for (int k = 1; k <=9; k++){
            if(checkValid(S,x,y,k)){
                S[x][y] = k;
                solveSudoku(S, x, y+1);
                S[x][y] = 0;
            }
        }
    } else {
        solveSudoku(S,x,y+1);
    }
}
```

```
boolean checkValid(int S[][9], int x, int y, int k){
    for(int i = 0; i <9 ; i++){
        if(S[x][i] == k) return false;
    }
    for(int i = 0; i <9 ; i++){
        if(S[i][y] == k) return false;
    }
    int a = x/3, b = y/3;
    for(int i = 3*a; i < 3*a+3; i++){
        for(int j = 3*b; j < 3*b+3; j++){
            if(S[i][j] == k) return false;
        }
    }
    return true;
}
```



83

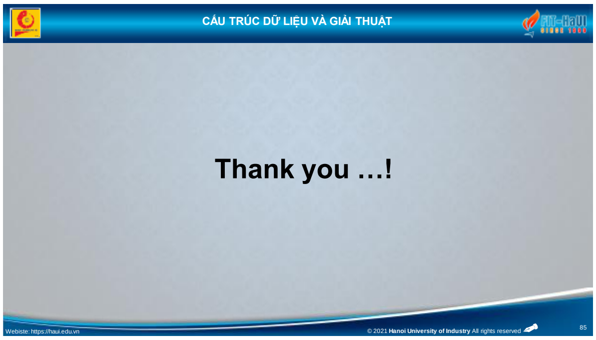
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

**Nhận xét:**

- ***Ưu điểm***: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.
- ***Nhược điểm***: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:
  - Rơi vào tình trạng "thrashing": qúa trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.
  - Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.
  - Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết đc nhánh tìm kiếm sẽ đi vào bế tắc.



84


CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
Thank you …!
Website: https://haui.edu.vn
© 2021 Hanoi University of Industry All rights reserved
85

85