# câu hỏi lý thuyết

**Ý nghĩa của thiết kế thuật toán**

Việc thiết kế thuật toán giúp cho việc hiểu biết về các thuật toán và các chiến lượt xây dựng thuật toán cho phép các lập trình viên các nhà khoa học máy tính có nền tảng lý thuyết vững chắc có nhiều sự lựa chọn hơn trong việc đưa ra các giải pháp cho các bài toán thực tế hay bài toán phức tạp.
hầu hết các chương trình được viết ra chạy trên máy tính dù lớn hay nhỏ dù đơn giản hay phức tạp đều phải sử dụng các cấu trúc dữ liệu tuân theo các trifh tự cách thức làm viêc nào đó chính là các giải thuật việc hiểu biết về các thuật toán và cá chiến lượt xây dựng thuật toán cho phép các lập trình viên các nhà khoa học máy tính có nền tảng lý thuyết vững chắc có nhiều sự lựa chọn hơn trong việc đưa ra các giải pháp cho các bài toán thực tế vì vậy việc học tập môn PTTKTT và đánh giá giải thuật là một điều quan trọng

"Thuật toán là một dãy hữu hạn các bước không mập mờ và có thể thực thi được, quá trình hành động theo các bước này phải dừng và cho kết quả mong muốn" (SGK)

thuật toán thỏa mãn 3 tiêu chuẩn Xác định + Hữu hạn + Đúng
Tính xác định/rõ ràng
- Tính đúng/chính xác
- Tính hữu hạn/dừng
- Tính đơn giản
- Tính khách quan
- Tính phổ dụng/tổng quát

▪ Tính hiệu quả: không gian và thời gian

● Một giải thuật được xem là tốt nếu đạt được các tiêu chuẩn sau:

- Tính đúng: chấp nhận các thuật giải đơn giản có thể cho kết quả đúng hay gần đúng nhưng có khả năng thành công cao hơn.
- Tính đơn giản
- Tính tối ưu (hiệu quả)

● Không gian (tốn ít bộ nhớ/tài nguyên máy tính)

● Thời gian chạy của thuật toán (thực hiện nhanh) • Trong khuôn khổ của môn học này, chúng ta chỉ quan tâm đến tiêu chuẩn thực hiện nhanh

## độ phức tạp của thuật toán

để kiểm tra tính đúng đắn của 1 giải thuật, ta thường sẽ phải thử nó với một bộ dữ liệu nào đó rồi so sánh kết quả thu được với kết quả đã biết. Tuy nhiên điều này không chắc chắn vì có thể giải thuật này đúng với bộ dữ liệu này nhưng lại không đúng với bộ dữ liệu khác. Tính đúng đắn của 1 giải thuật cần được chứng minh bằng toán, tạm thời chúng ta không đề cập ở đây.

Đối với các chương trình chỉ dùng 1 vài lần thì yêu cầu giải thuật đơn giản sẽ được ưu tiên vì chúng ta cần 1 giải thuật dễ hiểu, dễ cài đặt, ở đây không đề cao vấn đề thời gian chạy vì chúng ta chỉ chạy 1 vài lần.

Tuy nhiên, khi 1 chương trình sử dụng nhiều lần, yêu cầu tiết kiệm thời gian sẽ được đặc biệt ưu tiên. Tuy nhiên, thời gian thực hiện chương trình lại phụ thuộc vào rất nhiều yếu tố như: cấu hình máy tính, ngôn ngữ sử dụng, trình biên dịch, dữ liệu đầu vào, ... Do đó ta khi so sánh 2 giải thuật đã được implement, chưa chắc chương trình chạy nhanh hơn đã có giải thuật tốt hơn. "Độ phức tạp của thuật toán" sinh ra để giải quyết vấn đề này.

## So sách điểm giống và khác nhau giữa các chiến lượt thiết kế:

**Chia,giảm,biến đổi**

giống :các phương pháp đều biến đổi để chuyển sang một dạng dễ dẫn đến lời giải hơn để giải quyết bài toán

Khác:chia để trị :P(n) → p1,p2,p3 và n ≥2 chia thành các bài toán con đồng dạng , sau đó ta kết hợp nghiệm của các vấn đề con để nhận được nghiệm của vấn đề đã cho giảm để trị : quy bài toán gốc về bài toán con chỉ có 1 bài toán con để giải:

Biến đổi để trị: thay vì giải a → biến đổi cái b đã có lời giải thành lời giải của a(b khác hoàn toàn a) vd:

**Thuật toán chia để trị và quy hoạch động:**

Cả hai thuật toán đều hoạt động dựa trên nguyên tắc chia nhỏ bài toán lớn thành các bài toán nhỏ hơn để giải quyết. Từ đó đạt được đáp án mong muốn.

Tuy nhiên, nếu các bạn để ý thì sẽ thấy được hai điểm khác nhau khá rõ giữa chúng. Đầu tiên, chia để trị chỉ xử lý bài toán theo chiều từ trên xuống (top-down) trong khi quy hoạch động hỗ trợ cả hai chiều (top-down và bottom-up). Điểm thứ hai đó là việc chú trọng hiệu suất. Đối với thuật toán chia để trị, ta sẽ gọi đệ quy các bài toán con mà không quan tâm nhiều đến việc hạn chế xử lý các công việc tương tự nhau. Ngược lại, thuật toán quy hoạch động sẽ sử dụng một số kỹ thuật để tận dụng lại những gì có sẵn như memoization.

**phân biệt quy hoạch động vs tham lam:**

khác:

QHĐ: kế thừa và cả thiện nhược điểm của tham lam độ phức tạp tốt hơn (bé ) giải được bằng quy hoạc động mà kém kiệu quả (có cho kết quả chính xác 100% hay không ?) giống: quan tâm đến bật tăng trưởng hơn

**ưu nhược điểm của dùng toán học và thực nghiệm để để đánh giá tính hiểu quả về thời gian thực nghiệm**

ưu : đơn giản với n nhỏ với đối tượng giỏi lập trình ,để thực hiện thực nhiệm trực quan sẽ cụ thể hơn giúp người học có thể nắm vững đầy đủ các kiến thức hơn **nhược điểm:**

kém tin cậy (không có điều kiện thực nghiệm lý tưởng — không chắc thực nghiệm trên cùng máy tính hay cùng một người có cùng trình độ kiến thức) kết quả chỉ được công nhận bởi một cộng đồng nhỏ đếm bằng tay

Chịu sự hạn chế của ngôn ngữ lập trình.

▪ Ảnh hưởng bởi trình độ của người lập trình.

▪ Phụ thuộc vào phần cứng (cấu hình máy).

▪ Phải chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí. ▪ Bộ dữ liệu không thể đặc trưng hết

▪ Cần phải cài đặt chương trình và đo thời gian

**Toán học**

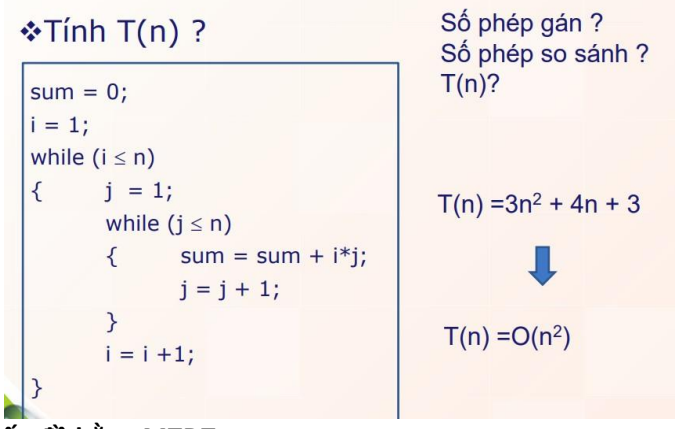
Đếm số lần "mỗi hoạt động" hoặc "hoạt động cơ bản" được thực hiện Hoạt động cơ bản: hoạt động quan trọng nhất, đóng góp nhiều nhất vào tổng thời gian chạy

Đơn vị đo thời gian thực hiện = số các lệnh được thực hiện trong một máy tính lý tưởng

▪ Thời gian = số phép gán + số phép so sánh (thông thường)

Thời gian thực hiện chương trình là một hàm phụ thuộc kích thước dữ liệu đầu vào

Ví dụ: Chương trình tính tổng của n số có thời gian thực hiện là T(n) = ? (đơn vị: số chỉ thị thực thi/ số phép toán ưu: kết quả có độ tin cậy cao hơn thông qua tính toán nhược điểm: yêu cầu có trình độ chuyên môn, khả năng vận dụng và kiến thức về toán học để chứng minh thuật toán




## Quy trình giải quyết vấn đề bằng MTĐT

1. Khảo sát, Phân tích
2. Lựa chọn phương pháp giải, ý tưởng chung

3. Thiết kế: Xây dựng mô hình dữ liệu, các bước xử lý
4. Mã hóa, viết CT: Cài đặt thành CTDL, đoạn CT cụ thể Diễn tả thuật toán theo NNLT đã chọn Chạy thử, kiểm tra:
5. Lỗi và cách sửa: Lỗi cú pháp, Lỗi ngữ nghĩa Xây dựng bộ dữ liệu test thực hiện, bảo trì, phát triển

Lý thuyết thiết kế thuật toán

**Define a problem**

1Tình huống, ngữ cảnh, Cơ sở dữ liệu – thông tin – tri thức của vấn đề (Base) 2. Giả thiết (Input)-3. Mục tiêu, yêu cầu (Output)- 4. Các điều kiện, ràng buộc, phạm vi **mô hình hóa vấn đề Vấn đề:** phải được đặc tả hay mô hình hóa dựa trên công cụ toán học hay các ngôn ngữ đặc tả(hình thức) cho máy tính ,Mô hình cho từng thành phần và tổng hợp lại →mô hình cho vấn đề

**Giếng ma thuật**

Mô tả bài toán: •Bạn đang đứng trước một cái giếng ma thuật, trên đó có ghi 2 số nguyên dương a và b. •Mỗi lần ném một viên sỏi xuống giếng, nó sẽ trả về cho bạn a*b đồng tiền vàng, sau đó a và b sẽ tăng lên 1. •Vậy nếu bạn có n viên sỏi thì sẽ kiếm được bao nhiêu đồng tiền vàng?

**Input** 2 :số nguyên a và b với $1 \le a,b \le 2000$

Số nguyên không âm n với $0 \le n \le 5$ là số viên sỏi mà bạn có được

**Output**:Một số nguyên cho biết số đồng tiền vàng kiếm được Ví dụ: Cho a = 1, b = 2 và n = 2, output là 8 đồng

**một số phương pháp thiết kê thuật toán**

**Phương pháp chia để trị**

(Divide and Conquer) Phương pháp chia để trị có tư tưởng là chia bài toán ban đầu thành các bài toán con tương tự

Các bài toán con tiếp tục được chia nhỏ hơn, cứ chia liên tiếp như vậy cho tới khi gặp bài toán con đã có lời giải hoặc có thể dễ dàng đưa ra lời giải

Sau đó lần lượt giải các bài toán con này và kết hợp các kết quả lại với nhau ta thu được kết quả cần tìm của bài toán ban đầu

**Phương pháp quay lui (Backtracking)**

Tư tưởng của thuật toán quay lui đó là tìm nghiệm của bài toán bằng cách xem xét tất cả các phương án có thể

Ta có thể thử duyệt các phương án cho đến khi tìm thấy phương án đúng còn gọi là phương pháp thử sai

Với tốc độ xử lí nhanh của máy vi tính thì phương pháp này có thể giải quyết được nhiều bài toán tuy nhiên nếu kích thước bài toán quá lớn thì nó trở nên không phù hợp

Bởi vì nếu kích thước bài 6 toán lớn thì kéo theo thời gian duyệt các phương án và độ phức tạp về mặt không gian cũng lớn và có thể lớn đến mức nào đó không thể chấp nhận được

Do đó phương pháp này thường chỉ hữu dụng với các bài toán có kích thước nhỏ

**Phương pháp quy hoạch động (Dynamic Programming )**

Phương pháp quy hoạch động nhìn chung được áp dụng cho lớp các bài toán tối ưu và cho kết quả đúng

Trong thuật toán giải bài toán tối ưu thường phải giải quyết nhiều trường hợp có thể quy về các bài toán con để giải và lựa chọn giải pháp tối ưu nhất của bài toán Các bước để giải bài toán bằng quy hoạch động gồm những công việc chính sau đây: – Tìm nghiệm của bài toán con nhỏ nhất (Thường là trường hợp suy biến); – Tìm công thức (gọi là công thức truy hồi) xây dựng nghiệm cho bài toán con thông qua nghiệm của bài toán con nhỏ hơn; – Tạo bảng phương án lưu trữ giá trị nghiệm của các bài toán con; – Từ bảng phương án suy ra nghiệm của bài toán ban đầu cần giải

**Phương pháp tham lam (Greedy Method)**

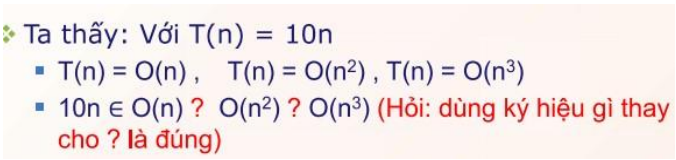
Tư tưởng của phương pháp tham lam như sau: Ta xây dựng dần dần các thành phần của nghiệm

Giả sử ta đã xây dựng được k thành phần của véc tơ nghiệm ( x1 , x 2 , x k ) , nhiệm vụ tiếp theo là phải xây dựng được thành phần thứ k+1 tức là tìm xk+1 Lúc này thành phần xk+1 được lựa chọn theo tiêu chí là tốt nhất theo hàm f(X) trong tập các ứng cử viên Sk để được ( x1 , x 2 , x k , x k+1 )

Cứ tiếp tục xây dựng như vậy cho đến hết các thành phần của nghiệm X Với cách làm như vậy có một số bài toán nếu xây dựng được hàm f(X) thích hợp thì có thể tìm được nghiệm tối ưu Còn đối với phần nhiều bài toán ta chỉ tìm được nghiệm gần đúng với nghiệm tối ưu khi sử dụng thuật toán tham lam

## ký hiệu tiềm cận

### 1 chứng minh bằng định nghĩa




dùng ký hiệu ∈

log(n) = n^c

dạng 1 : chứng minh tính chất

### 2 khẳng định là đúng sai

sai → chứng minh đúng → phát hiện mâu thuẫn → kết luận $n \le \log2(c)$ với mọi n > n0 vẽ hình ra và chứng minh không đúng → kết luận

3 sắp xếp theo order of growth (độ tăng trưởng)


các phép biến đổi nên lưu ý :**log2(n) = n^c**
**n =2^log2(n)**
**c=0.52**

exp:

f1 = tổng từ 1 đến n của i : o(n^2) f2 = n mũ logn = vì (logn > 2 ) do số mũ lớn hơn nên f1< f2 f3 = log(log(n)) + nlog(n) = o(nlog(n)) | log(n) = n^c | = o(n*n^c) = o(1+c) ⇒ f3<f1<f2 f4 = n ^ căn n = O(c) n = 2^(log2(n)) ⇒ 2^[log2(n)*căn n]

1,000001^n = 2^log2(1,0000001) = 2^(log2(1,00000001)*n) =2^O(n) f5= 2^20000 =O(c) f6= 1,0000001 = O(c^n) ⇒ f3<f1<f2 <f6

# phân tích thuật toán

## đệ quy đếm G(n) + ss(n)

```
sum =0;i=1 2g
while(i<=n)   n+1 ss
  j=1     1g
  while(j<=n)    n+1 ss
  {
    if(i==j)|| i+j == n+1) 2n ss
```

```
            sum=sum+aij   anpha i gán
          j++;        n g
        }
        i++  1g
```

gán(n) = 2 +2*n+ n^2 + n*ai tìm số th trùng của đk if :
i==j và i+j = n+1 ⇒ 2i =n+1 ⇒ nếu n lẽ có 1th trùng ⇒ ai = 2n-1
⇒ nếu n chẵn ⇒ ai = 2n ai = (2n - (n // 2)) ct rút gọn

so sánh (n) = n+1 n*2n + tổng 1 đến n của 2n - n trừ đi số lần sai của vt (i+j==n+1) đệ quy

## thành lập PTDQ giải pt đệ quy

Vd1:

```
public int g(int n) {
   if (n == 1)
      return 2;
   else
      return 3 * g(n / 2) + g( n / 2) + 5;
}
```

- Gọi $T(n)$ là thời gian thực hiện $g(n)$
- Khi $n = 1$, chương trình thực hiện việc duy nhất là return 2. Ta có $T(1) = C_1$

- Khi $n > 1$, chương trình phải gọi đệ quy 2 lần $g(n/2)$, tốn thời gian là $2T(\frac{n}{2})$, sau đó cộng tổng và trả về kết quả. Thời gian thực hiện các phép toán và trả về kết quả là một hằng số $C_2$.
- Ta có phương trình đệ quy:

$$T(n) = \begin{cases} C_1, & khi\ n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2, & khi\ \ n > 1 \end{cases}$$

Vd2:

```
long xn(int n)
{
  if (n == 0) return 1;
  long s = 0;
  for (int i=1; i<=n; i++)
    s = s + i*i*xn(n–i);
  return s;
}
```

- Gọi T(n) là thời gian thực hiện *xn(n)*.
- Khi n = 0, chương trình thực hiện duy nhất việc *return 1*. Ta có T(0) = $C_1$
- Khi n > 0, chương trình phải gọi đệ quy *xn(n-i)* n lần, nhân kết quả với với (i*i) và cộng tổng và return (với i chạy từ 1 tới n, bước tăng là 1). Thời gian thực hiện các phép toán và return là một hằng $C_2$.
  Ta có phương trình đệ quy:

$$T(n) = \begin{cases} C_1 & n = 0 \\ nT(n-1) + nC_2 & n > 0 \end{cases}$$

**Vd3:**

```
waste (n)
{   if (n==0) return 0;
    for (i = 1 to n )
        for (j = 1 to i )
            print i,j,n;
    for (i = 1 to 3)
            waste (n/2);
}
```

Gọi $T(n)$ là thời gian thực hiện $Draw(n)$

- Khi $n < 1$, chương trình thực hiện việc duy nhất là trả về kết quả là 0. Ta có $T(n) = C_1\ khi\ n < 1$.
- Khi $n \geq 1$, chương trình thực hiện vòng lặp for ngoài n lần, vòng lặp for trong $n^2$ lần, thời gian thực hiện 2 vòng lặp là $n^2C_2$, sau đó thực hiện lệnh print (“*”) tốn thời gian là một hằng $C_3$. Thời gian thực hiện 2 vòng lặp và lệnh print (“*”) là $n^2C_2 + C_3$.
- Sau đó chương trình thực hiện đệ quy $Draw(n-3)$ tốn thời gian là $T(n-3)$.
- Ta có phương trình đệ quy:

$$T(n) = \begin{cases} C_1, & khi\ n < 1 \\ T(n-3) + n^2C_2 + C_3, & khi\ \ n \geq 1 \end{cases}$$

## Phương pháp thay thế truy hồi

**T(n) = T(n-1) + 5  T(1) = 0**

$$\begin{aligned} T(n) &= T(n-1) + 5 \\ &= [T(n-2) + 5] + 5 = T(n-2) + 10 \\ &= [T(n-3) + 5] + 10 = T(n-3) + 15 \\ &= \cdots \\ T(n) &= T(n-i) + 5i \end{aligned}$$

Quá trình kết thúc khi $n - i = 1 \rightarrow i = n - 1$

$\rightarrow T(n) = T(1) + 5(n-1) = 5(n-1)$

$$\mathbf{T(n)} = \begin{cases} \mathbf{1} & \textbf{nếu n } = \mathbf{1} \\ \mathbf{2T}(\frac{\mathbf{n}}{\mathbf{2}}) + \mathbf{1} & \textbf{nếu n } > \mathbf{1} \end{cases}$$

T(n) = $2T(\frac{n}{2}) + 1$

= $2[2T(\frac{n}{4}) + 1\ ] + 1$ = $4T(\frac{n}{4}) + (1 + 2)$

= $4[2T(\frac{n}{8}) + 1\ ] + 2$ = $8T(\frac{n}{8}) + (1 + 2 + 4)$

Tại bước thứ i: T(n) = $2^i\ T(\frac{n}{2^i}) + 2^i - 1$

Quá trình trên kết thúc khi $\frac{n}{2^i} = 1$ => $2^i$ = n => i = $\log_2 n$

⇨ T(n) = n*T(1) + n-1

= 2n - 1

$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + n^2}$ **T(1) = 1**

$= 2 \cdot \left[2 \cdot T\left(\frac{n}{4}\right) + (\frac{n}{2})^2\right] + n^2$

$= 4 \cdot T\left(\frac{n}{4}\right) + \frac{n^2}{2} + n^2$

$= 4 \cdot \left[2 \cdot T\left(\frac{n}{8}\right) + (\frac{n}{4})^2\right] + \frac{n^2}{2} + n^2$

$= 8T\left(\frac{n}{8}\right) + \frac{n^2}{4} + \frac{n^2}{2} + n^2$

$= 2^i \cdot T\left(\frac{n}{2^i}\right) + n^2 \sum_{k=0}^{i-1} \frac{1}{2}^k$

Ta có: $\frac{n}{2^i} = 1 \Leftrightarrow i = log_2 n$

$\Rightarrow T(n) = 2^{log_2 n} \cdot T(1) + n^2 . (\frac{\frac{1}{2}^{log_2 n} - 1}{-0.5})$

$= n - 2n^2 . (\frac{1}{2}^{log_2 n} - 1)$

$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + \log n,\ \ T(1) = 1}$

$T(n) = 2T\left(\frac{n}{2}\right) + \log n$

$= 2[2T\left(\frac{n}{4}\right) + \log\left(\frac{n}{2}\right)\ ] + \log n = 4T\left(\frac{n}{4}\right) + 2\log\left(\frac{n}{2}\right) + \log n$

$= 8T\left(\frac{n}{8}\right) + \log n + 2\log\frac{n}{2} + 4\log\frac{n}{4}$

$= 2^i T\left(\frac{n}{2^i}\right) + \sum_{k=0}^{i-1}(2^k \log(\frac{n}{2^k}))$

$= 2^i T\left(\frac{n}{2^i}\right) + \log n \sum_{k=0}^{i-1} 2^k - \log 2 \sum_{k=0}^{i-1} k2^k$

$= 2^i T\left(\frac{n}{2^i}\right) + \log n\left(2^i - 1\right) - \log 2\left[(i-2)2^i + 2\right]$

$$T(n) = \begin{cases} 1, nếu\ n = 1 \\ 2^i T\left(\frac{n}{2^i}\right) + \log n\left(2^i - 1\right) - \log 2\left[(i-2)2^i + 2\right], nếu\ n > 1 \end{cases}$$

Quá trình kết thúc khi $\frac{n}{2^i} = 1 \Leftrightarrow i = \log_2 n$

Khi đó:

$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \log n\left(2^{\log_2 n} - 1\right) - \log 2\left[(\log_2 n - 2)2^{\log_2 n} + 2\right]$

$T(n) = n + (n-1)\log n - \log 2\,(n\log_2 n - 2n + 2)$

$T(n) = n + n\log n - \log n - n\log n + 2n + 2$

$T(n) = 3n - \log n - 2$

$\boldsymbol{. T(n) = 2T\left(\frac{n}{2}\right) + n^2}$ **T(1) = 1**

$= 2 \cdot \left[2 \cdot T\left(\frac{n}{4}\right) + (\frac{n}{2})^2\right] + n^2$

$= 4 \cdot T\left(\frac{n}{4}\right) + \frac{n^2}{2} + n^2$

$= 4 \cdot \left[2 \cdot T\left(\frac{n}{8}\right) + (\frac{n}{4})^2\right] + \frac{n^2}{2} + n^2$

$= 8T\left(\frac{n}{8}\right) + \frac{n^2}{4} + \frac{n^2}{2} + n^2$

$= 2^i \cdot T\left(\frac{n}{2^i}\right) + n^2 \sum_{k=0}^{i-1} \frac{1}{2}^k$

Ta có: $\frac{n}{2^i} = 1 \Leftrightarrow i = log_2 n$

$\Rightarrow T(n) = 2^{log_2 n} \cdot T(1) + n^2.(\frac{\frac{1}{2}^{log_2 n} - 1}{-0.5})$

$= n - 2n^2.(\frac{1}{2}^{log_2 n} - 1)$

**5. $T(n) = 2T(\sqrt{n}) + 1$**

Ta có:

$T(n) = 2T(\sqrt{n}) + 1$

$= 2\left[2T\left(\sqrt{\sqrt{n}}\right) + 1\right] + 1 = 4T\left(n^{\frac{1}{4}}\right) + (1+2)$

$= 4\left[2T\left(n^{\frac{1}{8}}\right) + 1\right] + (1+2) = 8T\left(n^{\frac{1}{8}}\right) + (1+2+4)$

…

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

Ta có:

$$T(n) = \begin{cases} 0, khi\ n = 2 \\ 2^i T\left(n^{\frac{1}{2^i}}\right) + \sum_{k=0}^{i-1} 2^k, khi\ n > 2 \end{cases}$$

Quá trình kết thúc khi $n^{\frac{1}{2^i}} = 2 \Leftrightarrow i = \log_2(\log_2 n)$

Khi đó:

$T(n) = 2^{\log_2(\log_2 n)} T\left(n^{\frac{1}{2^{\log_2(\log_2 n)}}}\right) + \sum_{k=0}^{\log_2(\log_2 n)-1} 2^k$

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

$= \log_2 n * 0 + \log_2 n - 1$

$T(n) = \log_2 n - 1$

# Dùng phương trình đặc trưng

**a.T(n) = 4T(n-1) -3T(n - 2)  và  T(0) = 1 ,T(1) = 2**

Ta có :

T(n) - 4T(n-1) + 3T(n - 2) = 0

Đặt T(n) = $X^n$ đưa về dạng ẩn X :

⇨ $X^n - 4X^{n-1} + 3X^{n-2} = 0$

Phương trình đặc trưng : $X^2 - 4X + 3 = 0$

⇨ (X-3)(X-1) = 0

⇨ phương trình có 2 ngiệm đơn X=3 và X = 1

T(n) = $3^n C_1 + 1^n C_2$

Với T(0 ) = 1 => $C_1 + C_2 = 1$

Với T(1) = 2 => $3C_1 + C_2 = 2$

Giải hệ phương trình trên, ta dược : $C_1 = C_2 = \frac{1}{2}$

➔ Kết luận: T(n) = $\frac{1}{2}3^n + \frac{1}{2}$

***b)* $T(n) = 10T(n-1) - 33T(n-2) + 36T(n-3)$ và**
**T(0) = 0, T(1) = 1, T(2) = 7**

- Xét phương trình dạng:

$T(n) - 10T(n-1) + 33T(n-2) - 36T(n-3) = 0$

- Đặt , đưa (1) về dạng phương trình ẩn X:

$X^n - 10X^{n-1} + 33X^{n-2} - 36X^{n-3} = 0$

$\Leftrightarrow X^{n-3}(X^3 - 10X^2 + 33X - 36) = 0$

$\Leftrightarrow X^{n-3} = 0\ hoặc\ X^3 - 10X^2 + 33X - 36 = 0$ (2)

- Phương trình đặc trưng: $X^3 - 10X^2 + 33X - 36 = 0$ có 1 nghiệm đơn $X_1 = 4$ và một nghiệm kép $X_2 = 3$

$T(n) = C_1 4^n + C_2 3^n + C_3 n 3^n$

- Dựa vào T(0), T(1), T(2) để tìm tham số, ta có:
  - T(0) = 0 → $C_1 + C_2 = 0$
  - T(1) = 1 → $4C_1 + 3C_2 + 3C_3 = 1$
  - T(2) = 7 → $16C_1 + 9C_2 + 18C_3 = 7$

Giải hệ phương trình, ta được: $C_1 = 1$, $C_2 = -1$, $C_3 = 0$

⇨ Kết luận: $T(n) = 4^n - 3^n$

# hàm sinh

$$a_n = n \quad \leftrightarrow \quad f(x) = \sum_{n=0}^{\infty} (n) x^n = \frac{x}{(1-x)^2}$$

$$a_n = 1 \quad \leftrightarrow \quad f(x) = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

$$a_n = n+1 \quad \leftrightarrow \quad f(x) = \sum_{n=0}^{\infty} (n+1) x^n = \frac{1}{(1-x)^2}$$

$$a_n = -1 \quad \leftrightarrow \quad f(x) = \sum_{n=0}^{\infty} (-1) x^n = \frac{1}{(1+x)}$$

$T(n) = \begin{cases} 1 \; khi \; n = 0 \\ 2T(n-1) + 7 \; khi \; n > 0 \end{cases}$

Hàm sinh của dãy vô hạn $\{T(n)\}_0^{\infty}$ là:

$$f(x) = \sum_{n=0}^{\infty} T(n) x^n$$

$$f(x) = \sum_{n=1}^{\infty} [2T(n-1) + 7] x^n + T(0) x^0$$

$$f(x) = \sum_{n=1}^{\infty} [2T(n-1)] \, x^n + 7 \left[ \sum_{n=0}^{\infty} (x^n) - 1 \right] + 1^{(*)}$$

$Xét \sum_{n=1}^{\infty} 2T(n-1) \, x^n = 2x \sum_{n=1}^{\infty} T(n-1) \, x^{n-1} = 2x(f(x))$

$Thế \; vào \; ^{(*)}$, ta có:

$$f(x) = 2x\big(f(x)\big) + 7\left[\frac{1}{1-x} - 1\right] + 1$$

$$f(x) = 2x\big(f(x)\big) + \frac{7x}{1-x} + 1$$

$$(1-2x) f(x) = \frac{1+6x}{1-x}$$

$$f(x) = \frac{1+6x}{(1-x)(1-2x)}$$

Đưa phương trình về dạng: $f(x) = \frac{A}{1-x} + \frac{B}{1-2x}$

Ta có: $A(1-2x) + B(1-x) = 1+6x \rightarrow \begin{cases} A+B=1 \\ -2A-B=6 \end{cases} \rightarrow \begin{cases} A=-7 \\ B=8 \end{cases}$

$$\Rightarrow f(x) = \frac{-7}{1-x} + \frac{8}{1-2x}$$

$$f(x) = -7 \sum_{n=0}^{\infty} (x)^n + 8 \sum_{n=0}^{\infty} (2x)^n$$

$f(x) = \sum_{n=0}^{\infty} [8(2)^n - 7] x^n$ mà $f(x) = \sum_{n=0}^{\infty} T(n) x^n$

$\Rightarrow$ T(n) = $8(2)^n - 7$

- b).

$\boldsymbol{T(n) = 7T(n-1) - 12T(n-2)}$ ***nếu n>=2***
$\boldsymbol{T(0) = 1}$
$\boldsymbol{T(1) = 2}$

Hàm sinh của dãy vô hạn $\{T(n)\}_0^{\infty}$ là:

$$f(x) = \sum_{n=0}^{\infty} T(n) x^n$$

$$= \sum_{n=2}^{\infty} [7T(n-1) - 12T(n-2)] x^n + 1 + 2x$$

$$= \sum_{n=2}^{\infty} [7T(n-1)] x^n - \sum_{n=2}^{\infty} [12T(n-2)] x^n + 1 + 2x$$

$Xét \; A = \sum_{n=2}^{\infty} [7T(n-1)] x^n = 7x \sum_{n=2}^{\infty} T(n-1) x^{n-1} = 7x[f(x) - 1]$

$Xét \; B = \sum_{n=2}^{\infty} [12T(n-2)] x^n = 12x^2 \sum_{n=2}^{\infty} T(n-2) x^{n-2} = 12x^2 . f(x)$

$\Rightarrow f(x) = 7x[f(x) - 1] - 12x^2 . f(x) + 1 + 2x$
$\Rightarrow f(x) - 7x . f(x) + 12x^2 . f(x) = -7x + 1 + 2x$

$\Rightarrow f(x) = \frac{1-5x}{1-7x+12x^2} = \frac{1-5x}{(1-3x)(1-4x)}$

$\Rightarrow f(x) = \frac{2}{1-3x} - \frac{1}{1-4x}$

$\Rightarrow f(x) = 2\sum_{n=0}^{\infty}(3x)^n - 1\sum_{n=0}^{\infty}(4x)^n$ mà $f(x) = \sum_{n=0}^{\infty} T(n)x^n$

$\Rightarrow T(n) = 2 * 3^n - 1 * 4^n$

❖ c).

T(n+1) = T(n) + 2(n+2) nếu n >=1
T(0) = 3

Thế n = n -1 ta được:

T(n) = T(n-1) +2(n+1)

Hàm sinh của dãy vô hạn $\{T(n)\}_0^{\infty}$ là:

f(x) = $\sum_{n=0}^{\infty}$ T(n) $x^n$

f(x) = $\sum_{n=1}^{\infty}[T(n-1) + 2(n+1)]x^n$ +T(0) $x^0$

f(x) = $\sum_{n=1}^{\infty} T(n-1)\,x^n$ + $2\sum_{n=1}^{\infty}(n+1)]x^n$ +3

Xét $\sum_{n=1}^{\infty} T(n-1)\,x^n$ = x $\sum_{n=1}^{\infty} T(n-1)\,x^{n-1}$

=x[f(x)]

⇨ f(x) = $xf(x)$+ $2\sum_{n=1}^{\infty}(n+1)]x^n$ +3

f(x) = $xf(x)$+ 2 $* \frac{1}{(1-x)^2}$ - 2 + 3 = $xf(x)$+ $\frac{2}{(1-x)^2}$ + 1

⇨ (1-x)f(x) = $\frac{2}{(1-x)^2} + 1$

⇨ f(x) = $\frac{2}{(1-x)^3} + \frac{1}{1-x}$

ta có $\frac{2}{(1-x)^3}$ = $\sum_{n=0}^{\infty}(n+1)^2\,x^n + \sum_{n=0}^{\infty}(n+1)\,x^n = \sum_{n=0}^{\infty}(n^2+3n+2)\,x^n$

⇨ f(x) = $\sum_{n=0}^{\infty}(n^2+3n+2)\,x^n + \sum_{n=0}^{\infty} x^n$

f(x) = $\sum_{n=0}^{\infty}[(n^2+3n+2)+1]\,x^n$

f(x) = $\sum_{n=0}^{\infty}[n^2+3n+3]\,x^n$

mà f(x) = $\sum_{n=0}^{\infty} T(n)x^n$

Vậy T(n) = $n^2 + 3n + 3$

# thiết kế thuật toán

## mô hình hóa

**gọi** x1,x2 là số lượng sản xuất được của kiểu mũ 1 và mũ 2

hàm mục tiêu: max z = x1*8 +x2*5

ràng buộc: x1 <= 150 x2 <= 200, 2x1+x2<=500 ,x1>=0 ,x2>=0

❖ Ví dụ:

Một công ty điện tử sản xuất 2 kiểu radio trên 2 dây chuyền độc lập. Công suất của dây chuyền 1 là 60 radio/ngày và dây chuyền 2 là 75 radio/ngày. Để sản xuất 1 chiếc radio kiểu 1 cần 10 linh kiện điện tử E và 1 chiếc radio kiểu 2 cần 8 linh kiện này. Số linh kiện này được cung cấp mỗi ngày không quá 800. Tiền lãi khi bán 1 radio kiểu 1 là 30USD và kiểu 2 là 20USD. Xác định phương án sản xuất cho lãi nhiều nhất trong ngày.

Giải bằng hình học và bằng phương pháp Fourier-Motzkin.

Đầu tiên phải mô hình hóa bài toán:

Gọi $x_1$, $x_2$ là số lượng radio kiểu 1 và kiểu 2 sản xuất trong 1 ngày
Hàm mục tiêu: max z = $30*x_1+20*x_2$
Các ràng buộc: $x_1 \le 60$, $x_2 \le 75$, $10*x_1 + 8*x_2 \le 800$, $x_1 \ge 0$, $x_2 \ge 0$.

❖Có dạng tổng quát:

Cho hàm f(X) là hàm mục tiêu xác định trên 1 tập hữu hạn các phần tử D (tập các phương án)

Mỗi X ∈ D có dạng X = (X1, X2, …, Xn) gọi là 1 phương án, Xi ∈ P (tập các biến)

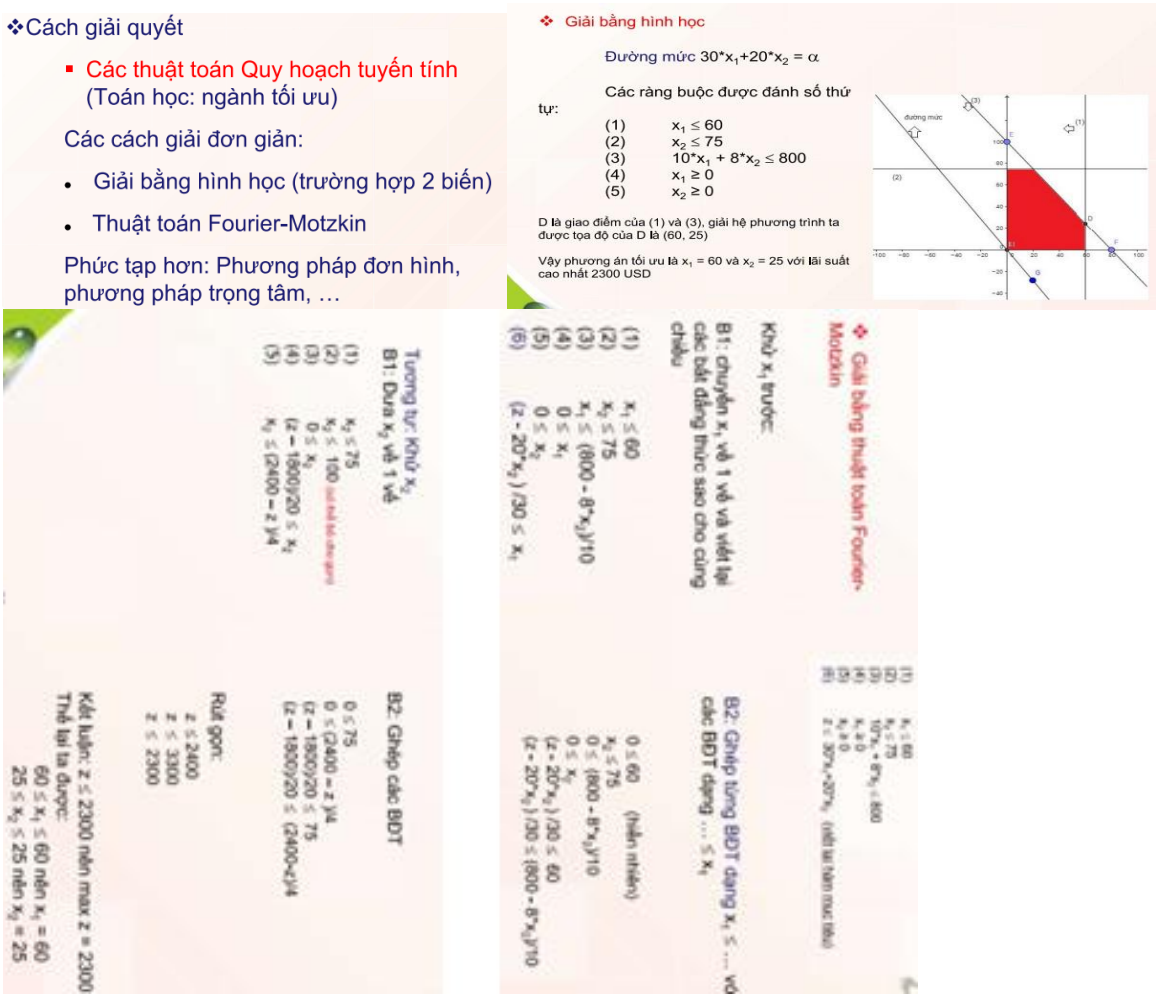
Cần tìm 1 phương án X "chấp nhận được" (thỏa mọi ràng buộc) sao cho f(X) đạt min (max) → phương án tối ưu

❖Cách giải quyết

- Vét cạn

Tuần tự xét tất cả các khả năng (phương án) có thể có cho đến khi gặp giải pháp cho vấn đề cần giải quyết→ thời gian mũ

- Các thuật toán Quy hoạch tuyến tính (Toán học: ngành tối ưu)
- Tối ưu cục bộ : phương pháp tham lam




# các chiến lượt thiết kế thuật toán

## tham lam

```
void freedy(P,n)
{
      G != 0
  #P là tập có khả năng được chọn
      while(P !=0)
      {
            x = chon(P) x là thành phần tối ưu
            p = p -{x} c cập nhập lại P (bỏ thành
phần x đã chọn)
            if( G ++ {x} (x chấp nhận đc ) )
                  G = G ++ {x}
      }
}
```

**vd 2 bài toán balo(fractional knapsack -dạng 2)**

B1 tính giá trị / kl (sau đó sắp xếp theo thứ tự giảm dần)
P[1...n] mảng chứa giá trị của n đồ vật đã được sắp xếp theo thứ tự tỉ lệ giảm dần ở B11
W[1....n]
X là vector lời giải Xi Є (0 <= Xi <= 1)
S tổng valua

```
void fractional knapsack(P,W,X,m(sức
chứa),n,con){
 con = m
      X = Ø
```

```
            #P tập đồ vật trong nhà
                for(i = 1->n)    {
                        if (W[i]) > con)
                                X[i] = (con/W[i])
                                S = S + X[i] * P[i]
                                break;
                        else
                                X[i] = 1;
                        con = con - w[i]
                }
            }
```

**bài toán tô màu:**

- mô hình hóa thành đồ thị

B1: chọn đỉnh chưa tô có bậc cao nhất(tham lam)

B2: Chọn lại màu đã sữ dụng (số màu sd ít nhất)

B3: sau khi tô hạ bậc các đỉnh kề và đánh dấu màu không được tô là màu hiện tại (mảng 2 chiều chứa đỉnh kề)

B4:

**vd1 bài toán giao hàng:**

```
void Giaohang(P,n){
        G != S (con đường giao hàng bắt đầu từ S)
 #P là tập đỉnh đồ thị trừ đỉnh bắt đầu {S}
        N=S
        while(P !=0)    {
                chọn đỉnh M trong P có khoảng cách tới n
nhỏ nhất
                cập nhập P =P\{M}
                cập nhập M vào G
                N = M
        }
```

Thêm S vào G động

vd 3: sắp xếp hoạt động

**sắp xếp hoạt động**

B1: xếp danh sách các hoạt động theo thứ tự fi tăng dần

F1<=F2<=3.... theo danh sách đã xắp xếp

S1<=S2<=S3...

```
void xeplich(P,n)
{
 G = {1};
        for(j = 2->n)
        {
                if(S[j]>=F[G[-1]])
                        G =G ⊍ {j}
        }
}
```

**B1: xếp danh sách các file theo thời lượng tăng dần**

F1<=F2<=F3.... theo danh sách đã xắp xếp

L1<=L2<=L3...

con =

```
void xeplich(P,n)
{

        }
}
```

phân công công việc

3 máy 10 công việc
phân công các công viwwjc cho mỗi mãy thời gian hoàn
thành toàn bộ cõgn việc là lớn nhất
t =(5,7,10,20,3,1,9,7,8,8)
xếp (20,10,9,8,8,7,)
vd:
M1 20
M2 10 8 7
M3 9  8 7
tgian lâu bố trí hco máy tthời gian làm việc  hiện có ít
nhất
nêu chưa ai làm việc gán cho máy đầu tiên

vector<int> ans //coong viec nao dc phan cho may nao
vector<pair<int,int>> Timejobs //thoigian job
vector<int> totaltime // tong thoi gian lam cua cac may
timmejobs .sort()

Lý thuyết:

- tiến hành qua nhiều bước ! quay lui → chọn đại ! tham lam → chọn trên tiêu chí tốt Chất 1 cấu trúc con tối ưu

lời giải toàn cục là tập hợp lời giải của các bài toán tối ưu cục bộ

- Tính Chất 1 cấu trúc con tối ưu

  lời giải được cấu trúc từ nhiều thanh phần x1,x2...

  chọn phương án tối ưu nhất (tính chất tham lam)

- TC 2 chọn phương thức tối nhất cho mỗi lời giải

Khuyết điểm : không chắc chắn cho lời giải chính xác

Thường cho lời giải tốt gần đúng chấp nhận được nhưng chưa hẳn là tối ưu

đôi khi trả về lời giải cực kỳ tệ

rất khó chứng minh tính đúng - chứng minh đc ⇒ thuật toán PP tham làm chỉ là thuật giải

## quay lui

**Câu hỏi định hướng**

Cách biểu diễn bàn cờ?
Các khả năng lựa chọn cho S (lựa chọn bước đi của con mã)
Điều kiện “chấp nhận được" của khả năng j
Cách thức xác định S theo khả năng j) khả năng j (cách lưu lại
sự lựa chọn
Cách ghi nhận trạng thái mới, trả lại trạng thái cũ của bài toán
Cách lưu hành trình của con mã (ghi nhận lời giải S)
**cây khung prim** Cây khung nhỏ nhất
Thuật toán Prim
Bắt đầu bằng việc chọn một đỉnh bất kỳ, đặt nó vào cây khung T.
Trong khi cây khung T có ít hơn n đỉnh
Ghép vào T cạnh có trọng số nhỏ nhất liên thuộc với một đỉnh của T và không tạo ra chu trình trong T.
note:Thuật toán dừng lại khi Tcó đủ n đỉnh hay (n-1) cạnh.
Có nhiều hơn một cây khung nhỏ nhất ứng với một đồ thị liên thông có trọng số.

- Thuật toán Prim
  - Bước 1: Khởi tạo
    - $V_T = \{s\}$; $E_T = \emptyset$; ($V_T$ – tập đỉnh; $E_T$ – tập cạnh)
    - $d_s = 0$; $v \notin V_T$ $d_v = w(s, v)$, nếu s và v liền kề
      $d_v = \infty$, nếu s và v không liền kề
  - Bước 2: Tìm cạnh
    - Tìm u mà $d_u = \min \{d_v \mid v \notin V_T\}$
    - $V_T = V_T \cup \{u\}$;
    - $E_T = E_T \cup \{e\}$ , e – cạnh nối u với một đỉnh của cây có trọng số $d_u$
    - Nếu $V_T \equiv V$ thì dừng.
  - Bước 3: Cập nhật nhãn
    - $d_v = \min \{d_v, w(u, v)\}$ với $v \notin V_T$

Thuật toán Kruskal

Bắt đầu bằng việc chọn một cạnh có trọng số nhỏ nhất, đặt nó vào cây khung T.

Trong khi cây khung T có ít hơn (n-1) cạnh

Ghép vào T cạnh có trọng số nhỏ nhất và không tạo ra chu trình trong T.

- Thuật toán Kruskal
  - Bước 1:
    - Sắp xếp các cạnh của đồ thị G theo thứ tự có trọng số không giảm: $w(e_1) \le w(e_2) \le \ldots \le w(e_m)$
    - $E_T = \{e_1\}$ , i =1
  - Bước 2: Tìm k = min { j | $E_T \cup \{e_j\}$ không có chu trình}
    $E_T = E_T \cup \{e_k\}$
  - Bước 3: i = i +1
    - Nếu i = n-1 thì dừng
    - Nếu i < n-1 thì quay lại bước 2

Mẫu thuật toán quay lui

```
try(i)
{
  for(j=1;->m;j++)
        if(chọn cạnh thêm vào và nếu hợp lệ)
        thêm lời giải vào
      if( i là bước giải cuối)
      {
        nếu lời giải mới có trọng số < min đã có thì lưu lời giải mới
        }
      else try(i+1)
  trả lại trạng thái đã thay đổi
}
```

# 3 Chia để trị :

**vd 4 bài toán vạch thước**

```
void vachthuoc(int L,int h,int l,int r)
{
mid = (l+r)/2
    if(l-r ==1 ) //1 mm
            return;
    vachthuoc(L/2,h-1,l,mid)
vachthuoc(L/2,h-1,mid+1,r)
draw(i);
}
```

**vd nhân 2 so nguyên lớn**

x*y = AC *10***n +[(A-B)(D-C) +AC +BD]*10**n/2 +BD)

```
int mul(x,y,n)
{
  A= left(x) C =left(y)
  B = right(x) D =right(y)
  if(n == 4)
          return x*y;

  k1 =a =a + mul(A,C,n/2);
  k2 =mul(A,D,n/2)
  k3 =mul(C,B,n/2)
  k4= mul(B,D,n/2)
```

```
    return k1*10*n + (k2+k3)**10^n/2 + k4
  }
  do phuc tap T(n) = 4T(n/2) + c1 +c2n;
```

**bai toán sắp hạng 2D**

sắp xếp các điểm theo hoành độ tăng dần ⇒ S{A,B,C,D,E}

B1 chia theo hoành độ thành 2 tập < mid và > mid

B2: duyệt điểm bên tập > mid nếu tung độ lớn hơn ++ rank chi phan tu do

MANG KET QUA R[i] ={0}

```
void rank(S,n,l,t)
{
    if(l==r || n==0)
            reuturn;
mid = (l+r)/2;
    Rank(S,n/2,l,mid)
    Rank(S,n/2,mid+1,r)
for(i =mid+1;i<=r;i++)
    for(j= 0 ;j<=mid;j++)
            if(S[i]>s[j]) //tung do lon hon
                        R[i]++;
}
```

**bai toan tim tat ca hoan vi**

```
  void hoanvi(S)
  {
    if(S.size() ==1)
                ans.push_back(S[0]);
            hoanvi(S-1)
    for(int i =0 ;i<ans.size();i++)
            ans.pusback(ans[i].add(S[-1]))
            S.pusback(S[-1].add(ans[i]))
  }
  LG = s[0]
  bottopup
  {
    for(i =1;i<n;i++)
            temp;
            for(x in LG)
                {
                    temp.push_back(insert(x,LG[i]))
                    temp.push_back(insert(LG[i],x))
                }
            LG =temp;
  }

  void tapcon(S)
  {
    if(S.size() ==0)
                ans.push_back(S[0],{rong});
    tapcon(S-1)
    ans.push_back(S[-1])
    for(int i =0 ;i<ans.size();i++)
            ans.pusback(ans[i].add(S[-1]))
```

S.pusback(S[-1].add(ans[i]))

## 4 quy hoạch động

— Các bước giải bài toán quy hoạch động:
- + Mô hình hóa được các trạng thái của bài toán.
- + Xác định sự tương quan giữa các trạng thái (Công thức truy hồi).
- + Giải bài toán cơ sở.
- + Xây dựng bảng phương án.
- + Truy xuất kết quả.

trình bày:

nhân chuỗi ma trận

A1,A2….An tìm cách nhân chi phí ít nhất b1 phân tích đặc trưng optima

substructure cm: kết luận:

lời giải = (A1.A2.A3).(A4.A5) đi tìm lời giải tối ưu thì ta phải tìm cách đặt ngoặc tối ưu của bài toán con 1 và bài toán con 2 tìm cách đặc ngoặc tối ưu và chi phí cost A1,A2…An = cost(a1,a2…ai) +cùng

⇒ lơi giải tối ưu được cấu thành từ lời giải tối ưu của bài toán con

B2 lập pt đệ quy và tạo bảng chi phí phép nhân cuối cùng :

po*pi*pn gọi m[i,j] là chi phí tối ưu của cost(ai…aj) gọi m[1,n] là chi phí tối ưu của cost(a1…an)

M[1,n] =max(m[1,1] + m[2,n] + chi phí cuối cùng = P0*P1*Pn pt quy hoạch động m[i,j] = max(m[i,k]+m[k+1,j] + pi-1*pk*pj )

B4 tra cứu và xuy ngược lời giải

**quy hoạch động cho bài toán knapsack**

Cho 4 đồ vật và một cái ba lô có thể đựng trọng lượng tối đa 10, mỗi đồ vật $i$ có trọng lượng $w_i$ và giá trị là $p_i$.

Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá $M$ và tổng giá trị là lớn nhất. Mỗi đồ vật hoặc là lấy đi hoặc là bỏ lại.

**1) Phân tích đặc trưng "Optimal substructure";**

Với mỗi đồ vật nếu theo giải thuật vét cạn để tìm ra cách chọn các đồ vật sao cho tốt ưu thì độ phức tạp sẽ là O(2n )

Ta nhận thấy giải pháp tối ưu của bài toán knapsack trên với trọng lượng tối đa là 10 có thể được xây dựng từ giải pháp tối ưu của bài toán con tối ưu ví dụ :

- Bài toán này có Optimal substructure

Ta nhận thấy rằng: Giá trị của cái túi phụ thuộc vào 2 yếu tố: Có bao nhiêu vật đang được xét và trọng lượng cái túi có thể chứa được, do vậy chúng ta có 2 đại lượng biến thiên. Cho nên hàm mục tiêu sẽ phụ thuộc vào hai đại lượng biến thiên. Do vậy bảng phương án của chúng ta sẽ là bảng 2 chiều F[i,m] là một cấu trúc tối ưu cho bài toán này.

Mỗi trạng thái:Trong đó:

: giai đoạn đưa ra quyết định chọn hay không chọn vật thứ : sức chứa của túi trước khi đưa ra quyết định.

: giá trị lớn nhất của chiếc túi sau khi đưa ra quyết định với sức chứa của túi = m

Mối quan hệ giữa các trạng thái là : **2) Xác định phương trình quy hoạch động** Phương trình quy hoạch động:

$$F_{i,m} = \begin{cases} 0 \text{ nếu } i = 0 \text{ or } m = 0 \\ max(F_{i-1,m},\ v_i + F_{i-1,m-w_i}) \end{cases}$$

**3) Tạo bảng lưu trữ kết quả của các bài toán con khi giải lần đầu;**

Tạo bảng có i+1 dòng và m+1 cột để lưu trữ kết quả của các bài toán con ,có F[0,j] = 0

4) **Xây dựng lời giải của bài toán ban đầu;**

Khởi tạo mảng kết quả ans[] có i + 1 phần tử đều =0 (chưa chọn)
Khởi tạo phần tử D = F[I,m] = phần tử cuối cùng trong bảng lưu trữ kết quả
For( k := i- 1 đến 0)
Nếu D == 0 : dừng vòng lặp
Nếu D == F[k,m] : (đồ vật thứ k+1 không được chọn) continue
Nếu D != F[k,m] :(đồ vật thứ k+1 được chọn)ó ans[k+1] =1
D=F[k,m] , m-=w[k+1]

5) **Minh họa áp dụng bằng cách điền giá trị vào bảng và truy xuất lời giải theo Ví dụ bên dưới** Truy xuất lời giải: D = F[4,10]=18 :

ta có D = F[4,10] =18 != F[3,10] = 12 => vật thứ 4 được chọn ,D = 12,m = 8 xét phần tử tiếp theo là D = 12 != F[2,8]= 4 => vật thứ 3 được chọn,D =4 ,m =5 xét phần tử tiếp theo là D = 4 == F[2,8] => vật thứ 2 không được chọn
xét phần tử tiếp theo là D = 4 != F[0,5] = 0 => vật thứ 1 được chọn D =0,m = 1
D == 0: dừng
Các đồ vật cho vào túi là: 1, 3, 4 và tổng giá trị là 18

**Prefix & Suffix calculation**

```
prefix[]
prefix(n)
        if n == 0 :
                pref[n] = 0
        else
                pref[n] = prefix(n - 1) + a[n]
        return pref[n]
botom up
prefix(n):
        pref[0] = 0
        for (i = 1; i <= n; i += 1):
                pref[i] = pref[i - 1] + a[i]
        return pref[n]
query(l, r):    return pref[r] - pref[l - 1]
```

— Phân tích **Dynamic Programming on Grid** lai xe chi phi thap

+ Trạng thái: Vị trí của xe được gọi là trạng thái
+ Hàm mục tiêu: Chi phí thời gian để xe tới và vượt qua được một trạng thái nào đó (tức là tới một nào đó và đi qua khỏi ô đó) được gọi là giá trị hàm mục tiêu
+ Điều khiển: Từ một trạng thái xe phải di chuyển tới trạng thái mới, việc lựa chọn một cách di chuyển được gọi là điều khiển
+ Trạng thái đầu S: Xe ở ngoài vùng chướng ngại chuẩn bị vào
+ Trạng thái kết thúc F: Xe đã qua vùng chướng ngại và đang ở vùng này

**Longest Increasing Subsequence dãy con tăng dần dài nhất**

```
def LIS() -quy hoach dong
    Initialize array d which has n + 1 elements;
    d[1] = 1;
    for (i: 2 -> n)
    {
            lc = 0;
            for (j: 1 -> i - 1)
                    if (A[j]<A[i] && d[j]>d[lc])
                            lc = j;
            d[i] = d[lc] + 1;
    }
    return max_element(d);
```

```
def LIS()-trace
    Initialize array d which has n + 1 elements;
    Initialize array previous which has n + 1 elements;
    d[1] = 1;
    for (i: 2 -> n)
    {
            lc = 0;
            for (j: 1 -> i - 1)
                    if (A[j]<A[i] && d[j]>d[lc])
                            previous[i] = j;
                            lc = j;
            d[i] = d[lc] + 1;
    }
    return max_element(d);
```

truy vet ket qua

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | 1 | 3 | 18 | 9 | 6 | 2 | 7 | 15 | 2 | 10 | 13 | 3 |
| $d$ | 1 | 2 | 3 | 3 | 3 | 2 | 4 | 5 | 1 | 5 | 6 | 3 |
| $previous$ | 0 | 1 | 2 | 2 | 2 | 1 | 5 | 7 | 1 | 7 | 10 | 6 |