TOÁN RỜI RẠC VÀ THUẬT TOÁN

(DISCRETE MATHEMATIC AND ALGORITHMS)

Bài 3

Một số phương pháp thiết kế thuật toán cơ bản

(Fundamental methods for designing algorithm)

Nguyễn Thị Hồng Minh

minhnth@hus.edu.vn

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

TOÁN RỜI RẠC VÀ THUẬT TOÁN

Bài 2

Một số phương pháp thiết kế thuật toán cơ bản

(*Fundamental methods for designing algorithm*)

Nguyễn Thị Hồng Minh
minhnth@gmail.com

Nội dung

- **Kĩ thuật đệ quy**
- **Phương pháp chia để trị**
- **Phương pháp quay lui**
- **Phương pháp nhánh cận**
- **Phương pháp quy hoạch động**

Chú ý: Hầu hết các hình vẽ trong các bài giảng được sưu tầm từ internet và được trình bày theo quan điểm của giảng viên.

ĐỆ QUY

RECURSIVE

KĨ THUẬT ĐỆ QUY

- **Khái niệm**
- **Lược đồ giải thuật**
- **Chứng minh tính đúng**
- **Đánh giá độ phức tạp**

Khái niệm về giải thuật đệ quy

- **Giải thuật đệ quy**:
  - Nếu một bài toán T được thực hiện bằng giải thuật của một bài toán T' có dạng giống như T.
  - Tính dừng:
    - T' phải "đơn giản hơn" T;
    - T' giải được (không cần đệ quy) trong một số trường hợp nào đó (suy biến).
- **Đặc trưng của các bài toán có thể giải bằng đệ quy**
  - Các bài toán phụ thuộc tham số;
  - Ứng với các giá trị đặc biệt nào đó của tham số thì bài toán có giải thuật để giải (trường hợp suy biến)
  - Trong trường hợp tổng quát bài toán có thể quy về dạng tương tự với một bộ giá trị mới của tham số và sau hữu hạn lần sẽ dẫn tới trường hợp suy biến.

Lược đồ giải thuật đệ quy

- **Lược đồ**

```
Recursive_Algorithm(p_n) ≡
    if (p_n = p_0)    //TH suy biến
        <Thực hiện giải thuật trường hợp suy biến>;
    else    //TH tổng quát
        <Lệnh;>
        Recursive_Algorithm(p_n-1);
        <Lệnh;>
    endif
End.
```

$$p_n \leftarrow p_{n-1} \leftarrow p_{n-2} \leftarrow \cdots \leftarrow p_o$$

Lược đồ giải thuật đệ quy

- **Ví dụ**
  - **Tính n!**

```
S(n):int ≡      //Hàm đệ quy tính giá trị n giai thừa
        if (n==1)
            S = 1;
        else
            S = n * S(n-1)
 End.
```

Lược đồ giải thuật đệ quy

- **Ví dụ**
  - **Tháp Hà Nội**

```
ChuyenThap(n,A,B,C) ≡ //Thủ tục đệ quy chuyển tháp n tầng từ A sang B
    if (n==1)
        Chuyen1Tang(A,B);
    else{
        ChuyenThap(n-1,A,C,B);
        Chuyen1Tang(A,B);
        ChuyenThap(n-1,C,B,A);
    }
End.

Chuyen1Tang(A,B) ≡ //Thủ tục chuyển 1 tấng tháp từ A sang B
     print(A,"→",B)
End.
```

Độ phức tạp giải thuật đệ quy

- **Xác định quan hệ truy hồi trong phép đệ quy**
  - Gọi $T(n)$ là độ phức tạp của giải thuật đệ quy với kích thước bài toán n
  - c = const là độ phức tạp trong trường hợp suy biến khi $n=n_0$; $T(n_0) = c$
  - Với $f(n)$ là hàm biến đổi tham số n, nếu giải thuật được thực hiện a lần bài toán con với tham số $f(n)$ thì:

    $T(n) = a.T(f(n)) + g(n)$ với $g(n)$ là độ phức tạp các thao tác ngoài lời gọi đệ quy

  - Công thức truy hồi xác định độ phức tạp giải thuật đệ quy

$$T(n) = \begin{cases} c & \text{khi } n = n_0 \\ a.T(f(n)) + g(n) & \text{trường hợp ngược lại} \end{cases}$$

Độ phức tạp giải thuật đệ quy

- **Ví dụ tìm công thức truy hồi**
  - Thuật toán S(n)

$$T_S(n) = \begin{cases} 1 \quad \text{khi } n = 1 \\ T_S(n-1) + 1 \text{ khi n} > 1 \end{cases}$$

  - Thuật toán ChuyenThap(n,A,B,C)

$$T_{ChuyenThap}(n) = \begin{cases} 1 \quad \text{khi } n = 1 \\ 2T_{ChuyenThap}(n-1) + 1 \text{ khi n} > 1 \end{cases}$$

Độ phức tạp giải thuật đệ quy

- **Giải công thức truy hồi**
  - Sử dụng phép thế liên tiếp

$$T_S(n) = \begin{cases} 1 \quad \text{khi } n = 1 \\ T_S(n-1) + 1 \text{ khi n} > 1 \end{cases}$$

$$\begin{aligned} T_S(n) &= T_S(n-1) + 1 = T_S(n-2) + 1 + 1 \\ &= T_S(n-3) + 1 + 1 + 1 \\ &= \ldots \\ &= T_S(n-(n-1)) + \underbrace{1 + \ldots + 1}_{n-1 \text{ lần n}} \\ &= 1 + 1 + \ldots + 1 = n \end{aligned}$$

$$\Rightarrow T_S(n) = O(n)$$

Độ phức tạp giải thuật đệ quy

- Giải công thức truy hồi
  - ***Định lí chính (General Theorem):*** Nếu n là lũy thừa của b thì công thức truy hồi:

$$T(n) = \begin{cases} 1 \quad \text{khi } n \le 1 \\ aT(n/b) + n^d \quad \text{khi } n > 1,\ a \ge 1, b > 1, d \ge 0 \end{cases}$$

Có lời giải là:

$$T(n) = \begin{cases} O(n^d) & \text{khi } a < b^d \\ O(n^d \log n) & \text{khi } a = b^d \\ O(n^{\log_b a}) & \text{khi } a > b^d \end{cases}$$

Chứng minh: Sử dụng phép thế liên tiếp.

Tham khảo Ian ParBerry's book, chương 4

Độ phức tạp giải thuật đệ quy

- Chứng minh định lí chính

$$T(n) = a^{\log_b n} T(n / b^{\log_b n}) + \ldots + a^2 (n / b^2)^d + a(n/b)^d + n^d$$

$$= n^{\log_b a} + n^d \sum_{j=0}^{\log_b n - 1} (a / b^d)^j$$

  - Trường hợp 1: $a < b^d$ $\quad T(n) < n^{\log_b a} + n^d \dfrac{1}{1 - (a/b^d)} = O(n^d)$
  - Trường hợp 1: $a = b^d$

$$T(n) = n^{\log_b a} + n^d \log_b n = O(n^d \log_b n)$$

  - Trường hợp 1: $a > b^d$

$$T(n) < n^{\log_b a} + n^d (a/b^d)^{\log_b n} \frac{1}{(a/b^d) - 1} = n^{\log_b a} \left(1 + \frac{1}{(a/b^d) - 1}\right) = O(n^{\log_b a})$$

Độ phức tạp giải thuật đệ quy

- Áp công thức truy hồi
  - $T(n) = 2T(n/2) + n^2$

    $a = 2, b = 2, d = 2, a < b^d \rightarrow T(n) = O(n^d) = O(n^2)$
  - $T(n) = 2T(n/2) + n$

    $a = 2, b = 2, d = 1, a = b^d \rightarrow T(n) = O(n^d \log_b n) = O(n \log n)$
  - $T(n) = 2T(n/2) + 1$

    $a = 2, b = 2, d = 0, a > b^d \rightarrow T(n) = O(n^{\log_b a}) = O(n)$

Độ phức tạp giải thuật đệ quy

- Áp dụng công thức truy hồi
  - $T(n) = 4T(n/2) + n^3$

    $a = 4, b = 2, d = 3, a < b^d \rightarrow T(n) = O(n^d) = O(n^3)$
  - $T(n) = 4T(n/2) + n^2$

    $a = 4, b = 2, d = 1, a = b^d \rightarrow T(n) = O(n^d \log n) = O(n^2 \log n)$
  - $T(n) = 4T(n/2) + n$

    $a = 4, b = 2, d = 1, a > b^d \rightarrow T(n) = O(n^{\log_b a}) = O(n^2)$

Chứng minh tính đúng giải thuật đệ quy

- Chứng minh bằng quy nạp
  - Thuật toán $S(n)$
    - Cơ sở quy nạp: $S(1) = 1$
    - Giả thiết quy nạp: $S(n) = n! = 1.2\ldots n$
    - Tổng quát: $S(n+1) = (n+1).S(n) = (n+1).1.2\ldots n = 1.2\ldots n.(n+1) = (n+1)!$
  - Thuật toán ChuyenThap(n,A,B,C)
    - Cơ sở quy nạp: ChuyenThap(1,A,B,C) = ChuyenTang(A,B)
    - Giả thiết quy nạp: ChuyenThap(n,A,B,C)
    - Tổng quát:

      ChuyenThap(n+1,A,B,C)=ChuyenThap(n,A,C,B); n tầng chuyển sang C

      ChuyenTang(A,B); 1 tầng chuyển sang B

      ChuyenThap(n,C,B,A) n tầng chuyển sang B

      => B có n+1 tầng => đúng

PHƯƠNG PHÁP CHIA ĐỂ TRỊ

DEVIDE AND CONQUER

PHƯƠNG PHÁP CHIA ĐỂ TRỊ

- **Giới thiệu**
- **Lược đồ giải thuật**
- **Chứng minh tính đúng**
- **Đánh giá độ phức tạp**
- **Ví dụ**

Giới thiệu

- **Ý tưởng giải thuật chia để trị**:

  Phương pháp thiết kế thuật toán dựa trên 2 thao tác chính:

  - Chia (*divide*): phân rã bài toán ban đầu thành các bài toán con có kích thước nhỏ hơn, có cùng cách giải.
  - Trị (*conque*): giải từng bài toán con (theo cách tương tự bài toán đầu - đệ qui) rồi tổng hợp các lời giải để nhận kết quả của bài toán ban đầu.

  Việc "Phân rã": thực hiện trên miền dữ liệu (chia miền dữ liệu thành các miền nhỏ hơn tương đương 1 bài toán con)

Mô hình và lược đồ giải thuật

- **Mô hình**

Xét bài toán P trên miền dữ liệu R.

Gọi D&C(R) là thuật giải P trên miền dữ liệu R.

Nếu R có thể phân rã thành n miền con ($R= R_1 \cup R_2 \cup \ldots \cup R_n$)

Với R_0 là miền đủ nhỏ để D&C(R_0) có lời giải thì lược đồ giải thuật chia để trị:

```
Divide&Conque(R)≡
    if(R=R0)
        Giải Divide&Conque(R0);
    else
        Chia miền R thành R1,R2,…,Rn
        for(i=1…n)
            Devide&Conque(Ri)
        Tổng hợp để nhận lời giải.
End.
```

Chứng minh tính đúng

- **Tính đúng của giải thuật chia để trị**
  - Chia để trị sử dụng kĩ thuật đệ qui, thông thường là đệ qui nhiều nhánh
  - Tính đúng của thuật toán Divide&Conque có thể được chứng minh như đối với giải thuật đệ qui sử dụng qui nạp.
    - Cơ sở quy nạp: Giải thuật ứng với $R=R_o$
    - Giả thiết quy nạp: Giải thuật đúng với các miền $R_i < R$
    - Tổng quát: Giải thuật đúng với miền $R=R_1 \cup R_2 \cup \ldots \cup R_n$

```
for(i=1…n)
        Devide&Conque(Ri)
Tổng hợp để nhận lời giải.
```

      - Chú ý: Chứng minh tính đúng của phần Tổng hợp để nhận lời giải, có thể phải sử dụng thêm các phương pháp khác.

Độ phức tạp thuật toán

- **Phân tích và đánh giá độ phức tạp thuật toán**
  - Xây dựng công thức truy hồi đánh giá độ phức tạp thuật toán
  - Giải công thức truy hồi xác định độ phức tạp thuật toán.
    - Phép thế liên tiếp
    - Sử dụng định lí chính

Thuật toán tìm kiếm nhị phân

- **Bài toán: Tìm kiếm nhị phân trên mảng được sắp**
  - Cho mảng *n* phần tử được sắp theo thứ tự (tăng dần) và một giá trị *x* bất kỳ. Kiểm tra xem phần tử *x* có trong dãy không ?
  - Phân tích ý tưởng: so sánh giá trị *x* với phần tử giữa của dãy tìm kiếm. Dựa vào giá trị này sẽ quyết định giới hạn tìm kiếm ở bước kế tiếp là nửa trước hay nửa sau dãy.

Thuật toán tìm kiếm nhị phân

- **Lược đồ thuật toán**

```
BinarySearch(a,x, L,R):int ≡
    //Tìm kiếm phần tử x trên dãy a từ vị trí L đến R
    if (L=R) return (x=a_L ? L : -1)
    else
        M = (L+R)/2;
        if (x = a_M) return (M);
        else
            if (x<a_M) BinarySearch(a,x,L,M)
            else BinarySearch(a,x,M+1,R)
        endif;
    endif;
End.
```

Thuật toán tìm kiếm nhị phân

- **Chứng minh tính đúng của thuật toán**

  Chứng minh qui nạp theo số phần tử của dãy

  - Cơ sở qui nạp: $n = R\text{-}L+1 = 1$ (dãy chỉ có 1 phần tử)
    - Câu lệnh `return (x=a`$_L$ `? L : -1)` trả lại giá trị L hoặc -1
  - Giả thiết qui nạp: Thuật toán đúng với mọi dãy có độ dài $n = R\text{-}L+1$
    - Tức là `BinarySearch(a,x, L,R)` trả về đúng kết quả tìm kiếm x với mọi dãy có độ dài $1 \le n' \le n = R\text{-}L+1$
  - Tổng quát: Chứng minh thuật toán đúng với $n+1 = R\text{-}L+2$
    - M=(L+R+1)/2; $L \le$ `M` $\le R$
    - Nếu $x=a_M$ thì kết quả trả về là M : đúng
    - Nếu $x<a_M$ thì kết quả là của bài toán $x \in a_L..a_M$? Theo giả thiết qui nạp thì `BinarySearch(a,x,L,M)` đúng vì $1 \le M\text{-}L+1=(R\text{-}L+1)/2+1 \le R\text{-}L+1$
    - Nếu $x>a_M$ tương tự

Thuật toán tìm kiếm nhị phân

- **Độ phức tạp thuật toán**

$$T(n) = \begin{cases} 1 \quad \text{khi } n = 1 \\ T(n/2) + 1 \text{ khi n} > 1 \end{cases}$$

$$\Rightarrow \mathrm{T}(n) = \mathrm{O}(\log n)$$

Thuật toán Merge Sort

Algorithm 3.1 Merge two lists

Pre-condition: $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_m$

$p_1 \longleftarrow 1$; $p_2 \longleftarrow 1$; $i \longleftarrow 1$

while $i \leq n + m$ **do**

if $a_{p_1} \leq b_{p_2}$ **then**

$c_i \longleftarrow a_{p_1}$

$p_1 \longleftarrow p_1 + 1$

else

$c_i \longleftarrow b_{p_2}$

$p_2 \longleftarrow p_2 + 1$

end if

$i \longleftarrow i + 1$

end while

Post-condition: $c_1 \leq c_2 \leq \cdots \leq c_{n+m}$

Thuật toán Merge Sort

Algorithm 3.2 Mergesort

Pre-condition: A list of integers $a_1, a_2, \ldots, a_n$

1: $L \longleftarrow a_1, a_2, \ldots, a_n$
2: **if** $|L| \leq 1$ **then**
3: **return** L
4: **else**
5: $L_1 \longleftarrow$ first $\lceil n/2 \rceil$ elements of L
6: $L_2 \longleftarrow$ last $\lfloor n/2 \rfloor$ elements of L
7: **return** Merge(Mergesort(L_1), Mergesort(L_2))
8: **end if**

Post-condition: $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

Thuật toán Merge Sort

- **Chứng minh tính đúng:** Sử dụng quy nạp
  - Cơ sở quy nạp: Giải thuật ứng với $R=R_o$
    - $|L| \leq 1$: Return L Dãy có 1 phần tử là dãy được sắp
  - Giả thiết quy nạp: Giải thuật đúng với các miền R_i mà $R_1 \cup R_2 \cup \ldots \cup R_n = R$
    - $L = L_1 \cup L_2$
    - Giải thuật đúng với miền L_1, L_2 tức là thuật toán Mergesort sắp xếp đúng hai dãy con L_1 và L_2
  - Tổng quát: Giải thuật đúng với miền R
    - Merge(Mergesort(L_1), Mergesort(L_2))
    - Tính đúng của thuật toán Merge: sử dụng bất biến vòng lặp

Thuật toán Merge Sort

- **Đánh giá độ phức tạp:**
  - Xây dựng công thức truy hồi:

$$T(n) = \begin{cases} 1 & \text{khi } n \le 1 \\ 2T(n/2) + n & \text{khi } n > 1. \end{cases}$$

  - Giải công thức truy hồi:

$$T(n) = 2T(n/2) + n$$

$$a = 2, b = 2, d = 1, a = b^d \rightarrow T(n) = O(n^d \log_b n) = O(n \log n)$$

PHƯƠNG PHÁP QUAY LUI

BACKTRACKING

PHƯƠNG PHÁP QUAY LUI

- **Ý tưởng**
- **Lược đồ giải thuật**
- **Chứng minh tính đúng**
- **Đánh giá độ phức tạp**
- **Ví dụ**

Ý tưởng

- Quay lui (Back tracking): Theo nguyên tắc vét cạn, nhưng chỉ xét những trường hợp "khả quan".
- Ví dụ: Thuật toán sinh dãy k_ary $A[1..n]$ có độ dài n
  - Vét cạn

```
k_ary(m) ≡ //Sinh dãy k-ary độ dài m
   if (m=0) process(A);
   else
      for j=0..k-1
         A[m]=j; k_ary(m-1);
End.
```

Ý tưởng

- Quay lui (Back tracking): Theo nguyên tắc vét cạn, nhưng chỉ xét những trường hợp "khả quan".
- Ví dụ: Thuật toán sinh dãy k_ary *A*[1..*n*] có độ dài *n*
  - Vét cạn chỉ xét trường hợp khả quan

```
k_ary(m) ≡ //Sinh dãy k-ary độ dài m
   if (m=0) process(A);
   else
      for j=0..k-1
         if (j là lựa chọn chấp nhận được cho A[m])
              A[m]=j; k_ary(m-1);
End.
```

Ý tưởng

- Quay lui:

  Tại mỗi bước, nếu có một lựa chọn được chấp thuận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì quay lại bước trước.

- Dùng để giải bài toán liệt kê các cấu hình:
  - Mỗi cấu hình được xây dựng bằng cách xác định từng phần tử
  - Mỗi phần tử được chọn bằng cách thử các khả năng ***có thể***.
  - Độ dài cấu hình tùy thuộc bài toán
    - Xác định trước: sinh dãy độ dài *n*
    - Không xác định trước: đường đi

Lược đồ giải thuật

- **Mô hình**

  Không gian nghiệm của bài toán (tập khả năng) X={(x_1,x_2,…,x_n)} gồm các cấu hình liệt kê có dạng (x_1, $x_{2, ...,}$ x_n) cần được xây dựng:

  - Cho x_1 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_1 thì:
  - Cho x_2 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_2 thì xét khả năng chọn x_3,…, x_n, => cấu hình tìm được (x_1, $x_{2, ...,}$ x_n).
  - ***Tổng quát:***

    Tại mỗi bước *i*: Xây dựng thành phần x_i

    - Xác định x_i theo khả năng *v*.
    - Nếu gặp điều kiện dừng (*i* = *n* hoặc x_i thỏa mãn điều kiện dừng) thì ta có được một lời giải, ngược lại thì tiến hành bước *i*+1 để xác định x_{i+1}.
    - Nếu không có một khả năng nào chấp nhận được cho x_i thì lại lùi lại bước trước để xác định lại thành phần x_{i-1} – sự quay lui

Lược đồ giải thuật

- **Lược đồ**

```
Try(i) ≡ //Sinh thành phần thứ i của cấu hình
    for (v thuộc tập khả năng thành phần nghiệm x_i)
        if ( x_i chấp nhận được giá trị v)
            x_i = v;
            <Ghi nhận trạng thái chấp nhận v>;
            if(gặp điều kiện dừng) //i=n hoặc x_i thỏa mãn đk dừng
                <xử-lí-nghiệm>;
            else        //lời gọi sinh thành phần tiếp theo của cấu hình
                Try (i + 1)
            <Khôi phục trạng thái chưa chấp nhận v>;
        endif;
    endfor;
End.
```

Lời gọi ban đầu: `Try(1)`

Chứng minh tính đúng

- **Sự hiển nhiên trong tính đúng của thuật toán quay lui**
  - Mỗi phần tử trong cấu hình nghiệm được chọn trong tập khả năng

```
for (v thuộc tập khả năng thành phần nghiệm x_i)
    if ( x_i chấp nhận được giá trị v)
       x_i = v;
```

  - Tính dừng của lời gọi đệ quy và cho ra kết quả đúng

```
if (gặp điều kiện dừng) //i=n hoặc x_i thỏa mãn đk dừng
      <xử-lí-nghiệm>;
else //lời gọi sinh thành phần tiếp theo của cấu hình
      Try (i + 1)
```

    - Đệ quy tiến Try(1) /x_1 → Try(2) /x_2 → … → Try(m) /x_m

      - Đủ thành phần nghiệm $m=n$ (n - hữu hạn)

      - Đến thành phần thỏa mãn điều kiện tức là x_m thỏa mãn điều kiện nghiệm

      Trường hợp không đạt đến thành phần thỏa mãn điều kiện nghiệm?

Đánh giá độ phức tạp

- **Công thức truy hồi**

$$T(n) = \begin{cases} 1 \quad \text{khi } n \le 1 \\ dT(n-1)+1 \quad \text{khi } n > 1. \end{cases}$$

d: max (hoặc giá trị trung bình) lực lượng của tập khả năng của các thành phần nghiệm x_i

Giải công thức truy hồi: $T(n)=O(d^n)$

- Là trường hợp xấu nhất (vét cạn)
- Trong triển khai thuật toán thời gian thực tế có thể giảm xuống

```
for (v thuộc tập khả năng thành phần nghiệm x_i)
    if ( x_i chấp nhận được giá trị v)
```

Ví dụ 1: Dãy k_ary

- **Bài toán:** Liệt kê dãy k_ary độ dài n
  - Phân tích:
    - Input: n, k
    - Output: Các dãy $X = (x_1 x_2 \cdots x_n)$ trong đó $x_i = 0,1,..k-1$
    - Dùng giải thuật Try(i) để sinh giá trị x_i
    - Nếu $i=n$ thì in giá trị nghiệm X, ngược lại sinh tiếp x_{i+1} bằng Try(i+1)
  - Lược đồ giải thuật:

```
Try(i) ≡
    for (v=0..k-1)   //v nhận giá trị từ 0 đến k-1
        xi = v;
        if (i=n) printResult (X);
        else Try(i+1) ;
    endfor;
End.
```

Lời gọi ban đầu **`Try(1)`**

Ví dụ 1: Dãy k_ary

- **Tính đúng:**
  - Mỗi x_i được chọn = $v \in \{0,..,k\text{-}1\}$
  - Đệ quy dừng khi chọn đủ n (hữu hạn) thành phần x_i trong một nghiệm X.

- **Độ phức tạp:**

$$T(n) = \begin{cases} 1 & \text{khi } n \le 1 \\ kT(n-1)+1 & \text{khi } n > 1. \end{cases}$$

Giải: $T(n) = O(k^n)$

Một số dạng bài toán quay lui

- **Quay lui sinh chuỗi:** Sinh chuỗi giá trị (độ dài có thể biết trước hoặc không), xác định các phần tử nghiệm theo phần tử trong chuỗi sinh.

  Ví dụ: binary string; k-ary string; knapsack problem; Travelling Salesperson Problem;…
- **Quay lui sinh hoán vị:** Sinh hoán vị của *n* phần tử, xác định thành phần nghiệm theo phần tử trong hoán vị sinh.

  Ví dụ: Hamilton cycle; Peaceful Queens;…
- **Quay lui sinh tổ hợp:** Sinh tổ hợp k phần tử từ n phần tử, xác định các phần tử nghiệm theo phần tử trong tổ hợp sinh.

  Ví dụ: clique problem; independent set problem; Ramsey number;…

PHƯƠNG PHÁP NHÁNH CẬN

BRANCH AND BOUND

Phương pháp nhánh cận

- **Mô hình**

  Không gian của bài toán (tập khả năng) $D=\{(x_1,x_2,\ldots,x_n)\}$ gồm các cấu hình liệt kê có dạng $(x_1, x_{2,\ldots\ldots,} x_n)$.

  Mỗi cấu hình x sẽ xác định một giá trị hàm chi phí $f(x)$

  $$f : D \to Z \qquad f(x) = C \ \ (C \in Z)$$

  Nghiệm của bài toán: $x= (x_1,x_2,\ldots,x_n)$ sao cho $f(x)$= *giá trị tối ưu (max/min)*

  Sử dụng lược đồ giải thuật quay lui:

  - Tại mỗi bước i: Xây dựng thành phần x_i
    - Xác định x_i theo khả năng *v*.
    - Tính chi phí lời giải nhận được. Nếu "*tốt hơn*" lời giải hiện thời thì chấp nhận x_i theo khả năng *v*. Tiếp tục xác định x_{i+1},…đến khi gặp nghiệm
    - Nếu không có một khả năng nào chấp nhận được cho x_i hoặc lời giải xấu hơn thì lùi lại bước trước để xác định lại thành phần x_{i-1}.

Phương pháp nhánh cận

- **Lược đồ**

```
Try(i,S )≡ //Sinh thành phần thứ i của cấu hình với chi phí hiện thời S
  for (v thuộc tập khả năng thành phần nghiệm x_i)
    if( v là chấp nhận được)
      T = S + Chi phí nghiệm khi có thêm thành phần v;
      if (T tốt hơn Toptimize) //Tốt hơn chi phí tốt nhất hiện có
        x_i  = v ;
        <Ghi nhận trạng thái chấp nhận v>;
        if( x_i là trạng thái kết thúc)
          <Ghi nhận nghiệm>;
          Toptimize = T; //Cập nhật chi phí tốt nhất
        else
          Try(i+1,T); //sinh thành phần tiếp theo với chi phí hiện thời T
        endif;
        <Khôi phục trạng thái chưa chấp nhận v>;
      endif;
    endif;
End.
```

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

DYNAMIC PROGRAMMING

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

- **Giới thiệu phương pháp**
- **Các bước xây dựng thuật toán**
  - Xác định cơ sở QHĐ, công thức truy hồi
  - Xây dựng bảng phương án
  - Tìm nghiệm tối ưu
  - Truy vết tìm nghiệm
- **Chứng minh tính đúng**
- **Đánh giá độ phức tạp**
- **Ví dụ**

Giới thiệu phương pháp

Bài toán đếm số tổ hợp

- Đếm số cách chọn *r* đồ vật từ *n* đồ vật cho trước
- Phân tích: chia để trị sử dụng đệ quy

$$C(r,n) = C(r\text{-}1,n\text{-}1) + C(r,n\text{-}1)$$

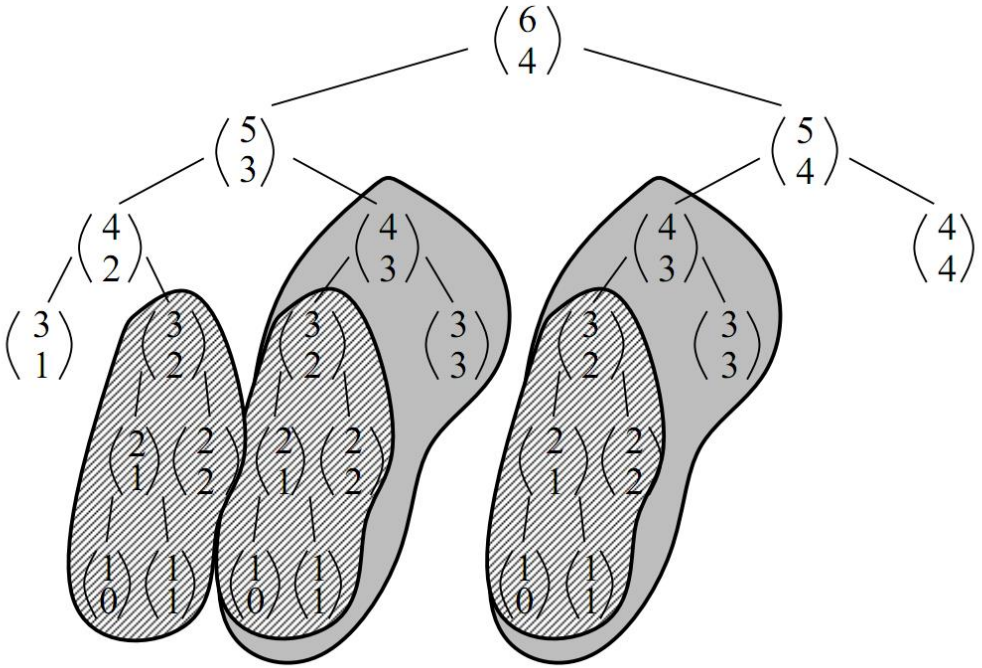
- Thuật toán

```
Choose(r,n) ≡ //Đếm số cách chọn r từ n
        if (r=0||n=r) return (1);
        else
            return(Choose(r-1,n-1)+ Choose(r,n-1);
End.
```

- Độ phức tạp: $T(n)=2T(n\text{-}1)+1 = O(2^n)$

Giới thiệu phương pháp

- **Bài toán đếm số tổ hợp**
  - Vấn đề: Nhiều bài toán con được giải trùng nhau!



Giới thiệu phương pháp

❖ Bài toán đếm số tổ hợp

- Giải pháp
  - Sử dụng bảng $C(i,j)$ để lưu các giá trị chọn i đồ vật từ j đồ vật

    $C(0,j) = 1$; $C(i,i) = 1$

    $C(i,j) = C(i-1,j-1) + C(i,j-1)$ $\quad i=1..r; j=1..n; i<=j$
  - Giá trị cần tìm là $C(r,n)$ – phần tử góc dưới, phải của bảng
  - Ví dụ bảng C với r=5, n=10

	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1	1
1		1	2	3	4	5	6	7	8	9	10
2			1	3	6	10	15	21	28	36	45
3				1	4	10	20	35	56	84	120
4					1	5	15	35	70	126	210
5						1	6	21	56	126	252

n=10

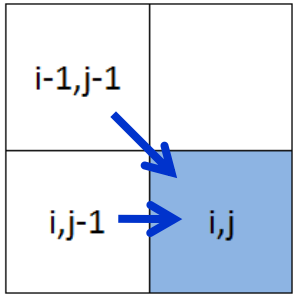
r=5

Giới thiệu phương pháp

❖ Bài toán đếm số tổ hợp

- Giải thuật:

```
Choose(r,n) ≡ //Đếm số cách chọn r từ n
    for i = 1..r C[i,i]=1;
    for i = 1..n-r C[0,i]=1;
    for i = 1..r
        for j = i+1..n-r+i
            C[i,j]=C[i-1,j-1]+C[i,j-1];
    return(C[r,n]);
End.
```



- Độ phức tạp:
  - Thời gian: $T(r,n)=(r+1)*(n-r+1)=nr+n-r^2+1 \le n(r+1)+1=O(n^2) << O(2^n)$
  - Không gian: Tối đa $O(nr)$. Tiết kiệm $O(r)$.

Giới thiệu phương pháp

- **Ý tưởng phương pháp qui hoạch động**
  - Qui hoạch động (Dynamic Programing-DP) là phương pháp giải các bài toán bằng cách kết hợp lời giải của các bài toán con theo nguyên tắc:
    - Giải tất cả các bài toán con (một lần)
    - Lưu lời giải của các bài toán vào một bảng
    - Phối hợp các bài toán con để nhận lời giải bài toán ban đầu.
  - Cách phát biểu khác: 1 bài toán giải bằng QHĐ được phân rã thành các bài toán con và bài toán lớn hơn sẽ được giải quyết thông qua các bài toán con này (bằng các phép truy hồi).
  - Phương pháp QHĐ thường dùng cho các bài toán tìm giá trị (hoặc giá trị tối ưu).

Giới thiệu phương pháp

- **So sánh chia để trị và QHĐ**
  - ***Chia để trị:***
    - Coi như các bài toán con đã có lời giải, qui bài toán lớn về các bài toán nhỏ hơn, cho tới khi gặp trường hợp suy biến.
    - Lời giải cho cùng 1 bài toán có thể bị lặp lại.
    - Phương thức Top-Down
  - ***Qui hoạch động:***
    - Giải trước tất cả các bài toán con rồi giải bài toán lớn dần
    - Kết quả của bài toán con (tối ưu) được lưu vào bảng phương án nên không bị giải lặp lại khi cần kết quả.
    - Phương thức Bottom-Up

Giới thiệu phương pháp

- **Nhận dạng bài toán giải bằng QHĐ**

  Các bài toán có các đặc trưng sau thì có thể giải bằng QHĐ

  - Bài toán có thể giải bằng đệ qui và tìm lời giải tối ưu.
  - Bài toán có thể phân rã thành nhiều bài toán con mà sự phối hợp lời giải của các bài toán con sẽ cho ta lời giải của bài toán ban đầu.
  - Quá trình tìm ra lời giải của bài toán ban đầu từ các bài toán con đơn giản hơn được thực hiện qua một số hữu hạn các bước có tính truy hồi.

Giới thiệu phương pháp

- **Điều kiện để thiết kế và thực hiện giải thuật QHĐ**
  - Câu hỏi: "Có phải mọi bài toán tối ưu đều có thể giải bằng phương pháp QHĐ hay không? "
  - Câu trả lời: "Nếu tìm được công thức truy hồi để giải bài toán lớn từ các bài toán nhỏ hơn thì sẽ giải bài toán được bằng QHĐ."

  => Vấn đề tìm công thức truy hồi là vấn đề quan trọng nhất khi giải một bài toán bằng phương pháp QHĐ.
  - Hơn nữa: Đủ không gian bộ nhớ cần thiết để lưu kết quả các bài toán con trong quá trình truy hồi.

Các bước giải bài toán bằng QHĐ

1. Nhận dạng bài toán giải bằng QHĐ
2. Xây dựng công thức truy hồi
3. Xác định và xây dựng cơ sở QHĐ
4. Dựng bảng phương án
5. Tìm kết quả tối ưu
6. Truy vết liệt kê thành phần nghiệm

Các bước giải bài toán bằng QHĐ

- **Nhận dạng bài toán giải bằng QHĐ**
  - Dựa vào dấu hiệu nhận dạng bài toán giải bằng QHĐ
  - Chú ý: không phải bài toán nào giải theo QHĐ cũng tốt hơn đệ qui, chẳng hạn như bài toán "Xếp ba lô 0-1" với số liệu lớn.
- **Xây dựng công thức truy hồi**
  - Đưa bài toán về 1 dạng cơ bản, và triển khai ý tưởng của dạng bài toán đó để nhanh chóng nhận ra hướng thiết lập công thức truy hồi.
  - Đây là bước khó nhất và cũng quan trọng nhất trong toàn bộ quá trình thiết kế thuật toán cho bài toán.
- **Xác định cơ sở QHĐ**
  - Dựa vào công thức truy hồi để nhận ra các bài toán cơ sở.
  - Dựa vào ý nghĩa của công thức truy hồi để thiết lập giá trị cho cơ sở.

Các bước giải bài toán bằng QHĐ

- **Dựng bảng phương án**
  - Dựa vào công thức truy hồi để tính giá trị các ô trong bảng phương án.
  - Chú ý: bảng phương án có thể 1 chiều, 2 chiều hoặc nhiều hơn
- **Tìm kết quả tối ưu**
  - Xác định vị trí chứa kết quả tối ưu của bài toán trên bảng phương án.
  - Chú ý: ngoài kết quả tối ưu, ô chứa kết quả tối ưu còn là điểm bắt đầu cho quá trình truy vết tìm nghiệm => lưu tọa độ của ô đó.
- **Truy vết liệt kê thành phần nghiệm**
  - Từ điểm bắt đầu là vị trí chứa kết quả tối ưu
  - Truy ngược lại về điểm bắt đầu của nghiệm: có thể là những ô đầu tiên trong bảng phương án (bài toán cơ sở), có thể là ô của bảng phương án đạt giá trị đầu.

Chứng minh tính đúng

- **Giai đoạn xây dựng bảng phương án**

```
for i = 1..r C[i,i]=1;
for i = 1..n-r C[0,i]=1;
for i = 1..r
    for j = i+1..n-r+i
        C[i,j]=C[i-1,j-1]+C[i,j-1];
```

  - Sử dụng quy nạp hoặc bất biến vòng lặp

- **Giai đoạn truy vết tìm nghiệm**
  - Sử dụng quy nạp
  - Sử dụng vòng lặp

Đánh giá độ phức tạp

- **Độ phức tạp thời gian**

  - Thời gian tính toán, xác định giá trị các phần tử trong bảng phương án, phụ thuộc vào số chiều của bảng phương án.

  $T(n) = O(n^k)$ - k số chiều của bảng phương án

  - Thời gian truy vết tìm nghiệm: Độ phức tạp thuật toán quay lui

- **Độ phức tạp không gian**

  - Kích thước bảng phương án – thường lớn

  - Giảm kích thước bảng phương án trong triển khai thuật toán

Ví dụ xây dựng công thức truy hồi

- ***Bài toán dãy con đơn điệu tăng dài nhất :*** Tìm dãy con dài nhất của một dãy đã cho. Các phần tử có thể không liên tiếp.

  Ví dụ: Dãy 2; 5; 4; 6; 3; 8; 9; 7

  => Dãy con dài nhất có 5 phần tử: 2; 5; 6; 8; 9

- **Phân tích:**
  - Input: (a,n)
  - Output: m – số lớn nhất các phần tử của dãy theo thứ tự tăng dần
  - Hàm tối ưu $L(i)$: Độ dài dãy con đơn điệu tăng dài nhất đến phần tử i

    Là độ dài các dãy con dài nhất đến j cộng 1 khi ghép thêm a_i vào sau, với điều kiện $j<i$, $a_j<a_i$
  - Công thức truy hồi: $L(i) = \max\{L(j)\}+1$ với $j<i$, $a_j<a_i$
  - Cơ sở QHĐ: $L(0) = 0$; $L(1) = 1$

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán dãy con đơn điệu tăng dài nhất***
    - Bảng phương án: Bảng 1 chiều L

i	0	1	2	3	4	5	6	7	8
a_i		2	5	4	6	3	8	9	7
L(i)	0	1							

    - Giá trị tối ưu: $\max(L(i))_{i=1..n} = 5$
    - Vậy dãy con đơn điệu tăng dài nhất có độ dài là 5
    - Vấn đề đặt ra: Dãy nào?

      - Từ giá trị tối ưu của bài toán tìm được, sử dụng bảng phương án đề tìm nghiệm của bài toán => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán dãy con đơn điệu tăng dài nhất***
    - Bảng phương án: Bảng 1 chiều L

i	0	1	2	3	4	5	6	7	8
a_i		2	5	4	6	3	8	9	7
L(i)	0	1	2						

    - Giá trị tối ưu: $\max(L(i))_{i=1..n} = 5$
    - Vậy dãy con đơn điệu tăng dài nhất có độ dài là 5
    - Vấn đề đặt ra: Dãy nào?

      - Từ giá trị tối ưu của bài toán tìm được, sử dụng bảng phương án đề tìm nghiệm của bài toán => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán dãy con đơn điệu tăng dài nhất***
    - Bảng phương án: Bảng 1 chiều L

i	0	1	2	3	4	5	6	7	8
a_i		2	5	4	6	3	8	9	7
L(i)	0	1	2	2					

    - Giá trị tối ưu: $\max(L(i))_{i=1..n} = 5$
    - Vậy dãy con đơn điệu tăng dài nhất có độ dài là 5
    - Vấn đề đặt ra: Dãy nào?

      - Từ giá trị tối ưu của bài toán tìm được, sử dụng bảng phương án đề tìm nghiệm của bài toán => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán dãy con đơn điệu tăng dài nhất***
    - Bảng phương án: Bảng 1 chiều L

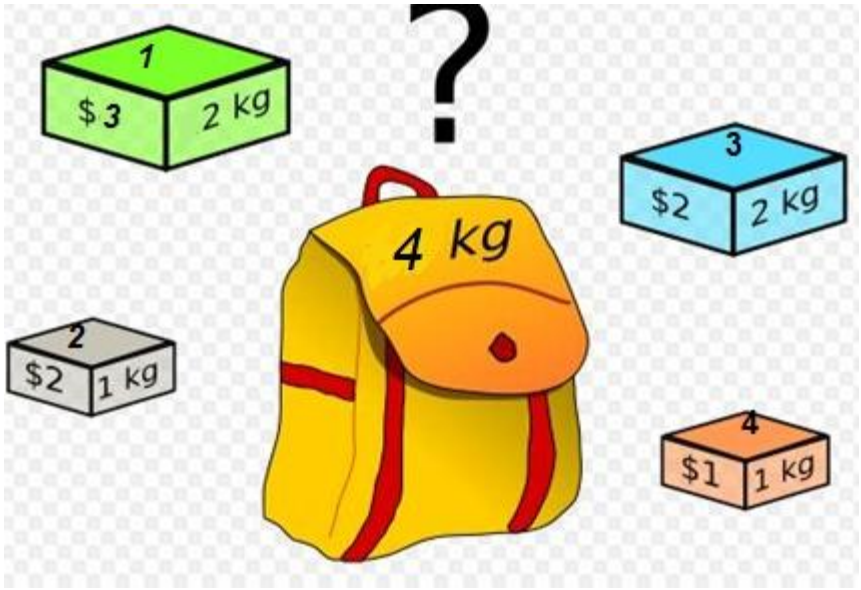
i	0	1	2	3	4	5	6	7	8
a_i		2	5	4	6	3	8	9	7
L(i)	0	1	2	2	3	2	4	5	4

    - Giá trị tối ưu: $\max(L(i))_{i=1..n} = 5$
    - Vậy dãy con đơn điệu tăng dài nhất có độ dài là 5
    - Vấn đề đặt ra: Dãy nào?

      - Từ giá trị tối ưu của bài toán tìm được, sử dụng bảng phương án đề tìm nghiệm của bài toán => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- ***Bài toán xếp balo 0-1:*** có N đồ vật với trọng lượng và giá trị tương ứng (p_i,w_i). Tìm cách cho các vật vào balo có trọng lượng P sao cho đạt giá trị cao nhất. Mỗi vật chỉ được chọn 1 lần (0-1).
  - Input: n, (p_i,w_i) i=1..n, P
  - Output: W= tổng giá trị lớn nhất các đồ vật cho vào balo



Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán xếp balo 0-1:***
    - Hàm tối ưu $C(i,j)$: Giá trị lớn nhất khi chọn các vật từ 1 đến i với trọng lượng balo j (i=0..n, j=0..W).

      - Nếu không chọn đồ vật thứ i thì: $C(i,j)= C(i\text{-}1,j)$

    - Nếu có chọn đồ vật thứ i thì $C(i,j) = C(i\text{-}1,j\text{-}p_i)+w_i$ (đkiện $p_i<=j$)
    - Công thức truy hồi

      $$C(i,j) = \max(C(i\text{-}1,j), C(i\text{-}1,j\text{-}p_i)+w_i)$$

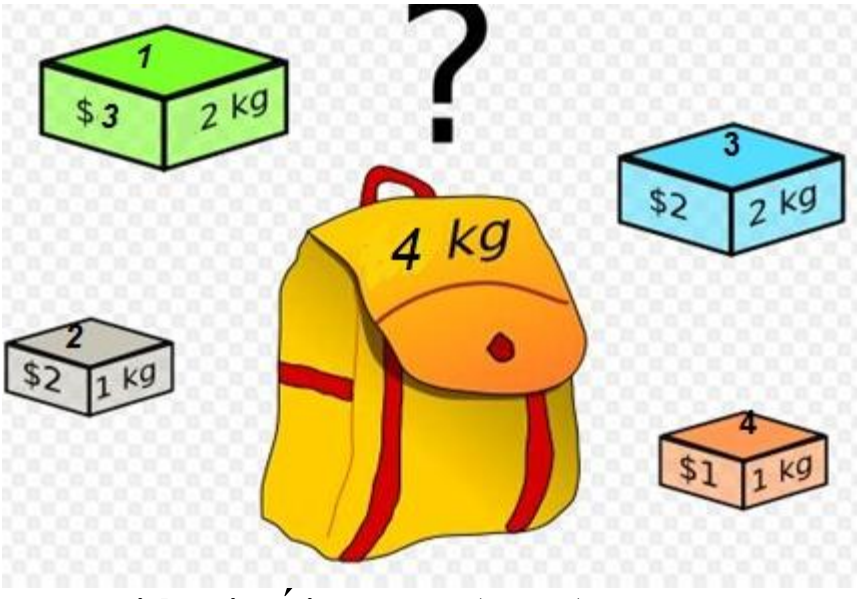
    - Cơ sở QHD

      $$C(i,0) = 0$$

      $$C(0,j) = 0$$

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán xếp balo 0-1:***
    - Bảng phương án: Bảng hai chiều lưu các giá trị $C(i,j)$



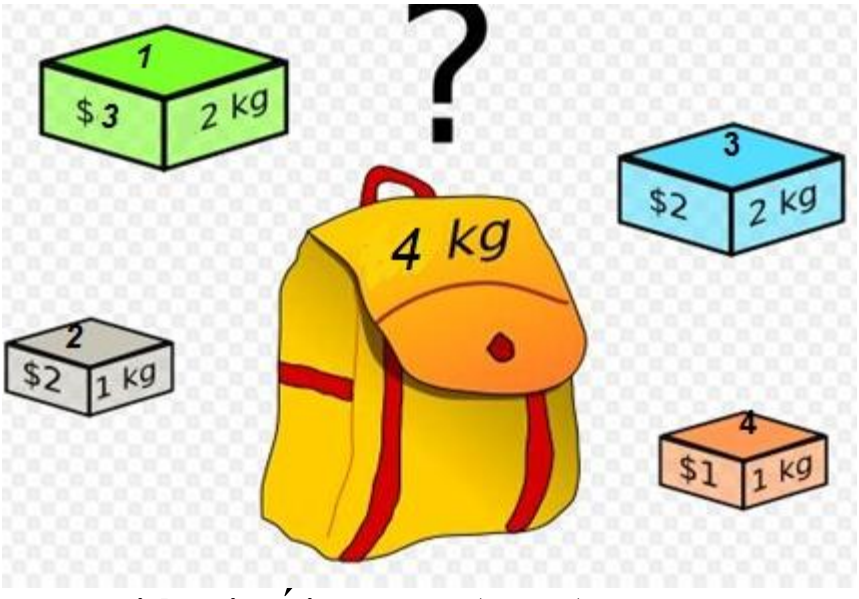
i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	0			
2	0	2			
3	0	2			
4	0	2			

$C(i,j) = \max(C(i-1,j), C(i-1,j-p_i)+w_i)$

    - Giá trị tối ưu: $C(n,W) = 6$
    - Những vật nào sẽ cho vào balo? => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán xếp balo 0-1:***
    - Bảng phương án: Bảng hai chiều lưu các giá trị $C(i,j)$



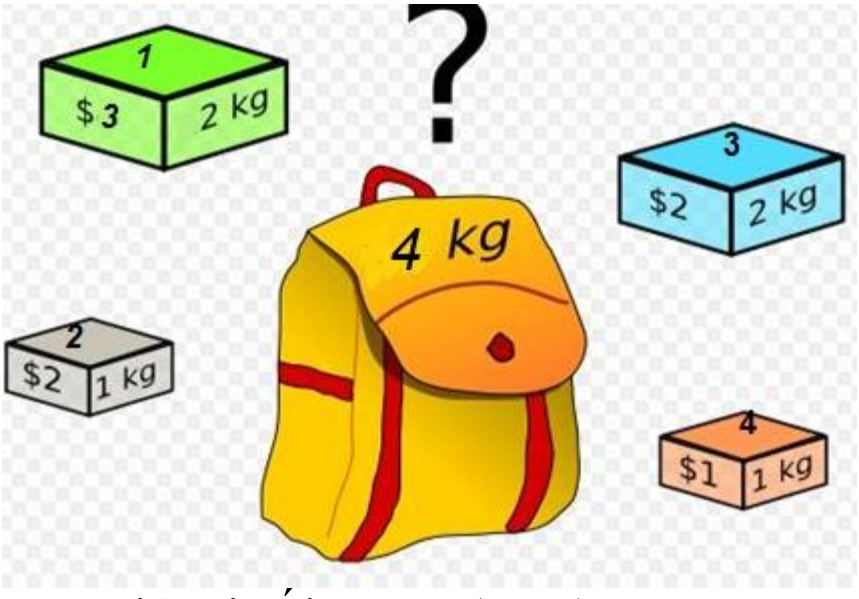
i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	0	3	3	3
2	0	2			
3	0	2			
4	0	2			

$C(i,j) = \max(C(i-1,j), C(i-1,j-p_i)+w_i)$

    - Giá trị tối ưu: $C(n,W) = 6$
    - Những vật nào sẽ cho vào balo? => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán xếp balo 0-1:***
    - Bảng phương án: Bảng hai chiều lưu các giá trị $C(i,j)$



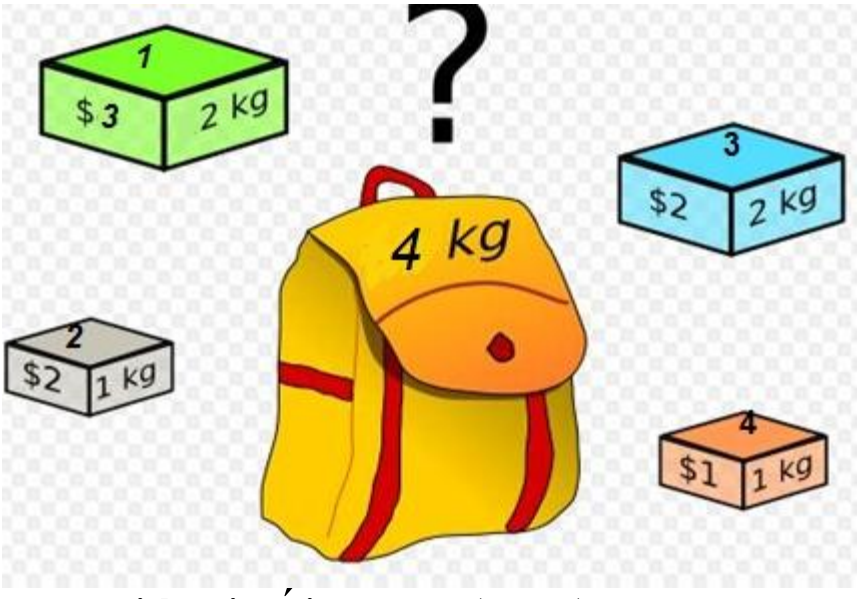
i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	0	3	3	3
2	0	2	3	5	5
3	0	2	3		
4	0	2	3		

$C(i,j) = \max(C(i-1,j), C(i-1,j-p_i)+w_i)$

    - Giá trị tối ưu: $C(n,W) = 6$
    - Những vật nào sẽ cho vào balo? => Truy vết tìm nghiệm

Ví dụ xây dựng công thức truy hồi

- **Ví dụ xây dựng công thức truy hồi và bảng phương án**
  - ***Bài toán xếp balo 0-1:***
    - Bảng phương án: Bảng hai chiều lưu các giá trị $C(i,j)$



i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	0	3	3	3
2	0	2	3	5	5
3	0	2	3	5	5
4	0	2	3	5	6

$C(i,j) = \max(C(i-1,j), C(i-1,j-p_i)+w_i)$

    - Giá trị tối ưu: $C(n,W) = 6$
    - Những vật nào sẽ cho vào balo? => Truy vết tìm nghiệm

Truy vết tìm nghiệm

- Dựa vào bảng phương án để dần dần tìm ra các thành phần nghiệm của bài toán.
- Điểm bắt đầu của quá trình truy vết: ô chứa giá trị tối ưu, kết quả của bài toán.
- Điểm kết thúc của quá trình truy vết: tùy thuộc bài toán, thường nằm ở những dòng, những cột đầu tiên của bảng phương án.

  - ***Chú ý:*** cùng 1 bài toán có thể có nhiều lời giải cùng cho 1 kết quả tối ưu. Việc truy vết bình thường chỉ đưa ra 1 phương án. Để hiển thị toàn bộ các nghiệm thì có thể dùng giải thuật đệ qui quay lui khi truy vết.

Truy vết tìm nghiệm

- **Truy vết tìm nghiệm**
  - ***Bài toán dãy con đơn điệu tăng dài nhất***

i	0	1	2	3	4	5	6	7	8
a_i		2	5	4	6	3	8	9	7
L(i)	0	1	2	2	3	2	4	5	4

- Bắt đầu từ vị trí tối ưu. $L(i)$ đạt max.
- Tại mỗi bước: Xét ô $L(i)$, phần tử đứng trước a_i trong dãy con là phần tử a_j nếu $j<i$, $a_j<a_i$ và $L(j)=L(i)-1$.
- Kết thúc khi đến phần tử đầu tiên của dãy con ($L(i) = 1$)
- Chú ý: Có thể có nhiều giá trị cùng thỏa mãn điều kiện này nhưng ta chỉ chọn 1 trong các phương án đó.

Truy vết tìm nghiệm

- **Truy vết tìm nghiệm**
  - ***Bài toán xếp balo 0-1***

i \ *j*	0	1	2	3	4
0	0	0	0	0	0
1	0	0	**3**	3	3
2	0	2	3	**5**	5
3	0	2	3	**5**	5
4	0	2	3	5	**6**

i	p_i	w_i	Chọn
0			
1	2	3	✓
2	1	2	✓
3	2	2	x
4	1	1	✓

- Bắt đầu từ ô $C(n,W)$

  - Nếu $C(i,j)= C(i\text{-}1,j)$ thì vật thứ i không được chọn và xét $C(i\text{-}1,j)$

  - Ngược lại $C(i,j) = C(i\text{-}1,j\text{-}p_i)+w_i$ thì vật thứ i được chọn và xét tiếp ô $C(i\text{-}1,j\text{-}p_i)$

- Kết thúc khi tới hàng 0 hoặc cột 0

Một số lỗi thường gặp

- **Lỗi trong khi thiết kế thuật toán**
  - **Nhận dạng không đúng bài toán**
  - **Xác định sai công thức truy hồi**
    - Xác định sai giá trị tối ưu cần tính
    - Nhầm lẫn giữa các cách kết hợp bài toán con
    - Xác định sai ý nghĩa của bảng phương án
  - **Xác định sai hoặc thiếu cơ sở QHĐ**
  - **Xác định sai kết quả tối ưu**
    - Muốn xác định được kết quả cần tìm nằm ở đâu trong bảng phương án phải dựa vào ý nghĩa của công thức truy hồi.
  - **Xác định sai điều kiện dừng khi truy vết**
    - Quá trình truy vết bắt đầu tại ô chứa kết quả tối ưu, nhưng kết thúc tại đâu còn phải dựa vào công thức truy hồi.

Một số lỗi thường gặp

- **Lỗi trong khi lập trình**
  - **Kích thước bảng phương án nhỏ hơn yêu cầu**
    - Bảng phương án có thể có thêm viền (hàng số 0, cột số 0) => bảng phương án cũng lớn hơn hoặc bằng kích thước miền dữ liệu.
  - **Không phải bảng phương án nào cũng tính theo từng hàng**
    - Việc hình thành dần kết quả của các bài toán con và lưu vào bảng phương án phải được thực hiện tuần tự nhưng tùy vào công thức truy hồi thì mới có thể biết được là tính bảng phương án theo từng hàng hay theo từng cột.
  - **Xét ô ở ngoài bảng**
    - Khi tính bảng phương án hoặc truy vết, cần chú ý các chỉ số để đảm bảo phần tử ở trong bảng.

Một số dạng bài toán QHĐ cơ bản

1. **Bài toán "Dãy con đơn điệu tăng dài nhất"**
2. **Bài toán "Xếp ba lô 0-1"**
3. **Bài toán "Đường đi trong bảng số"**
4. **Bài toán "Dãy con tổng bằng S"**

   Cho 1 tập có N số nguyên dương. Tìm 1 tập con có tổng bằng giá trị dương cho trước S.
5. **Bài toán "Biến đổi xâu"**

   Với 3 phép biến đổi xâu như sau:

   Chèn 1 kí tự vào sau vị trí i

   Xóa kí tự tại vị trí i

   Thay thế kí tự tại vị trí i bằng 1 kí tự khác

   Cho trước 2 xâu X, Y. Tìm cách biến đổi X về Y sao cho số phép biến đổi là ít nhất.

   Ví dụ: X = "aec"; Y = "bcd" Sử dụng ít nhất 3 phép biến đổi. Một cách là:

   - Chèn d vào sau vị trí thứ 3 (sau c); X="aecd"
   - Xóa kí tự ở vị trí thứ 2 (e); X="acd"
   - Thay kí tự ở vị trí số 1 (a) bằng kí tự b X="bcd"

Đưa một số bài toán về dạng cơ bản

1. **Bài toán "Dãy con chung dài nhất"**

   Cho 2 dãy số X (M phần tử) và Y(N phần tử)

   Tìm dãy con chung dài nhất của 2 dãy này

   - Dạng cơ bản: Biến đổi xâu

2. **Bài toán "Chia kẹo"**

   Cho N gói kẹo. Gói kẹo thứ i có A[i] cái kẹo.

   Chia các gọi kẹo thành 2 phần để chênh lệch giữa 2 phần là ít nhất

   - Dạng cơ bản: Dãy con tổng bằng S

3. **Bài toán "Lập lịch sử dụng tài nguyên"**

   Có *n* hoạt động (s_i,f_i) cùng đăng kí sử dụng một tài nguyên. Hãy sắp xếp trình tự để sao cho có nhiều tài nguyên được sử dung nhất.

   - Dạng cơ bản: Dãy con đơn điệu tăng dài nhất