ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - CƠ - TIN HỌC

BÁO CÁO

MÔN HỌC THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN

ĐỀ TÀI

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

DYNAMIC PROGRAMMING

Giảng viên hướng dẫn: PGS. TS. Nguyễn Thị Hồng Minh

Sinh viên: Đặng Trung Du - MSV: 18001108

Nguyễn Anh Thư - MSV: 18001212

Đại Học Quốc Gia Hà Nội, Trường Đại Học Khoa học Tự Nhiên, Khoa Toán – Cơ – Tin Học

Hà Nội, Ngày / /

Lời mở đầu

Trước tiên, chúng em xin dành lời cảm ơn sâu sắc với cô Nguyễn Thị Hồng Minh đã giúp đỡ và tạo điều kiện cho chúng em được tiếp xúc và tìm hiểu về môn Thiết kế và đánh giá thuật toán .

Dưới đây là bài tiểu luận của chúng em về việc tìm hiểu phương pháp quy hoạch động và một số ứng dụng của phương pháp.

Bài tiểu luận của chúng em còn nhiều thiếu sót, mong thầy cô và các bạn đóng góp ý kiến để chúng em có bài tiểu luận hoàn chỉnh hơn.

Mục lục

I) Giới thiệu chung

Quy hoạch động (Dynamic Programming – DP) là một phương pháp rất hiệu quả và quan trọng để giải các bài toán tin học, đặc biệt là những bài toán tối ưu. Quy hoạch động còn là một phương pháp làm giảm thời gian chạy của các thuật toán có tính chất của các bài toán con trùng nhau (overlapping subproblem) và cấu trúc con tối ưu (optimal substructure). Những bài toán này thường có nhiều nghiệm chấp nhận được và mỗi nghiệm có một giá trị đánh giá. Mục tiêu đặt ra là tìm nghiệm tối ưu, đó là nghiệm có giá trị lớn nhất hoặc nhỏ nhất (tối ưu). Ví dụ tìm đường đi ngắn nhất giữa 2 đỉnh đồ thị, tìm chuỗi con chung dài nhất của hai chuỗi, tìm chuỗi con tăng dài nhất,…

II) Quy hoạch động

1. Quy hoạch động là gì?

QHĐ giải các bài toán bằng cách kết hợp lời giải của các bài toán con của bài toán đang xét. Phương pháp này khả dụng khi các bài toán con không độc lập với nhau, tức là khi các bài toán con có dùng chung những bài toán "cháu". QHĐ giải các bài toán "cháu" dùng chung này 1 lần và lưu lời giải của chúng trong một bảng và không cần tính phải tính lại khi gặp lại bài toán đó.

QHĐ là một phương pháp phân tích và thiết kế thuật toán cho phép giảm bớt thời gian thực hiện khi khai thác tốt 2 đặc điểm nêu trên. Tuy nhiên thông thường QHĐ lại đòi hỏi nhiều không gian bộ nhớ hơn (để thực hiện ghi nhớ hay lưu trữ).

2. Một số đặc điểm của phương pháp QHĐ

Phương pháp QHĐ sử dụng nguyên lý bottom-up. Đầu tiên, ta giải các bài toán con đơn giản nhất, có thể tìm ngay ra nghiệm, sau đó kết hợp các bài toán con này lại để tìm lời giải cho bài toán lớn hơn, cứ như thế cho đến khi giải được bài toán yêu cầu. Với phương pháp này, mỗi bài toán con sau khi giải xong đều được lưu trữ lại và đem ra sử dụng nếu cần, do đó tiết kiệm bộ nhớ và cải thiện tốc độ.

Bài toán cần đến lập trình động khi có 2 đặc điểm sau đây:

Thứ nhất, bài toán thoả mãn nguyên lý tối ưu Bellman (nguyên lý này được thừa nhận mà không chứng minh: QHĐ là lớp các bài toán mà quyết định ở bước thứ i phụ thuộc vào quyết định ở các bước đã xử lý trước hoặc sau đó). Khi đó người ta nói bài toán có cấu trúc con tối ưu (optimal substructure), nghĩa là có thể sử dụng lời giải tối ưu của các bài toán con từ mức thấp để tìm dần lời giải tối ưu cho bài toán con ở các mức cao hơn, và cuối cùng là lời giải tối ưu cho bài toán toàn thể.

Thứ hai, bài toán có các bài toán con chồng/ gối lên nhau (overlapping subproblem). Nếu gọi 2 bài toán con cùng được sinh ra từ một bài toán là hai bài toán con cùng mức thì có thể mô tả hình ảnh các bài toán con phủ chồng lên nhau là: khi giải hai bài toán con cùng mức chúng có thể đòi hỏi cùng tham chiếu một số bài toán con thuộc mức dưới chúng. Bài toán con trùng nhau: bài toán lớn được chia ra làm các bài toán con, tuy nhiên các bài toán con được giải nhiều lần lại bởi các bài toán lớn khác nhau.

Giống với chia để trị, lời giải của bài toán quy hoạch động được tổ hợp từ lời giải các bài toán con. Tuy nhiên lại khác ở chỗ, thay vì gọi đệ quy, quy hoạch động sẽ tính trước lời giải của các bài toán con và lưu vào bộ nhớ (thường là dạng ma trận, hay còn được coi là dạng bảng), sau đó lấy lời giải của bài toán con ở trong mảng đã tính trước để giải bài toán lớn. Việc tìm nghiệm tối ưu của bài toán được thực hiện dựa trên việc tìm nghiệm tối ưu của các bài toán con. Kết quả của các bài toán con được ghi nhận lại để phục vụ cho việc giải các bài toán lớn hơn và giải được bài toán ban đầu. Chính việc lưu lại lời giải vào bộ nhớ như trên khiến cho ta không phải tính lại lời giải của các bài toán con mỗi khi cần, do đó, tiết kiệm được thời gian tính toán.

⇒ DP trở thành phương pháp làm giảm thời gian chạy của các thuật toán thể hiện các tính chất của bài toán con gối/ trùng nhau và cấu trúc con tối ưu, đồng thời còn tối ưu hoá trên phép đệ quy đơn giản.

3. Các yếu tố cơ bản của phương pháp

Phương pháp quy hoạch động giải quyết vấn đề theo nguyên tắc:

Step 1: Giải các bài toán con (bài toán con trùng/ gối lên nhau) một lần.

Step 2: Ghi lại vào bảng kết quả.

Step 3: Tổng hợp các kết quả được ghi lại để rút ra nghiệm.

Để thực hiện được các bước trên, ta cần xác định được các yếu tố cơ bản của phương pháp như sau:

a. Công thức truy hồi:

Là công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn hơn. Ta chia bài toán thành n giai đoạn và tối ưu dần từng giai đoạn từ 1- n. Tại giai đoạn thứ i: gọi S(i) là hàm số xác định giá trị tối ưu (max/min) cần xác định. Khi đã biết các giá trị tối ưu S(j) của các giai đoạn j từ 1 đến i – 1 thì ta có: S(i) = min/max{S(j) kết hợp với GTNN/ GTLN từ j đến i}, $j = 1 \div i - 1$.

b. Cơ sở QHĐ:

Là tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán con, ví dụ như các giá trị tối ưu có thể tính được ngay S(2), S(1), S(0). Cơ sở QHĐ thường tương ứng với trường hợp suy biến của thuật toán đệ quy.

c. Bảng kết quả/ bảng phương án:

Là không gian, bảng lưu trữ tất cả các giá trị tối ưu của các bài toán con để tìm cách phối hợp chúng. Bảng có thể là một mảng 1 chiều, 2 chiều hoặc nhiều hơn.

d. Kết quả tối ưu của bài toán:

Kết quả tối ưu của bài toán nằm trong bảng phương án, tuy nhiên vị trí của nghiệm bài toán trong bảng lại tuỳ thuộc vào từng bài toán do cách phối hợp kết quả của các bài toán để tìm kết quả cho bài toán lớn hơn là khác nhau (dựa vào công thức truy hồi).

e. Truy vết tìm nghiệm:

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

4. Các bước xây dựng thuật toán theo phương pháp QHĐ

Bao gồm 2 phần chính: *Thiết kế thuật toán* và *Triển khai thuật toán.*

Về *Thiết kế thuật toán*, gồm:

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

Để nhận dạng các bài toán có thể giải được bằng phương pháp QHĐ, ta có thể dựa vào các đặc điểm của bài toán giải bằng QHĐ như đã nói trên.

- **Xây dựng công thức truy hồi:**

Ta đưa bài toán về 1 dạng cơ bản, triển khai ý tưởng của dạng bài toán đó để nhanh chóng nhận ra hướng thiết lập công thức truy hồi. Hay có thể nói dựa vào nguyên lý tối ưu để tìm cách chia quá trình giải bài toán thành từng giai đoạn, sau đó tìm hệ thức biểu diễn tương quan giữa lời giải của bước đang xử lý với các bước đã xử lý trước đó. Hoặc tìm cách phân rã bài toán thành các bài toán con tương tự có kích thước nhỏ hơn, rồi tìm hệ thức nêu quan hệ giữa kết quả các bài toán con đó với bài toán đã cho dưới dạng hàm hoặc thủ tục đệ quy.

- **Xác định cơ sở QHĐ:**

Dựa vào công thức truy hồi để nhận ra các bài toán cơ sở hay dựa vào ý nghĩa của công thức truy hồi để thiết lập giá trị cho cơ sở.

Về *Triển khai thuật toán*:

- **Dựng bảng phương án:**

Dựa vào công thức truy hồi để tính giá trị cho các ô trong bảng phương án. Dữ liệu sẽ được tính toán dần theo các bước và được lưu trữ lại để giảm lượng tính toán lặp lại. Khi dựng bảng phương án, cần lưu ý chọn kích thước bảng hay miền nhớ lưu trữ dữ liệu càng nhỏ càng tốt, tương tự với kiểu dữ liệu, ta cũng cần phải chọn kiểu dữ liệu phù hợp, chọn đơn giản để dễ truy cập và xử lý.

- **Tìm kết quả tối ưu – nghiệm bài toán:**

Sau khi bảng phương án được xây dựng xong, ta cần xác định vị trí chứa kết quả tối ưu của bài toán trên bảng phương án. Ngoài việc thu được kết quả tối ưu, ô chứa kết quá tối ưu cũng quan trọng vì nó còn là điểm bắt đầu cho quá trình truy vết tìm nghiệm, do đó cũng cần phải ghi nhớ toạ độ của thành phần này.

- **Truy vết, liệt kê thành phần nghiệm:**

Từ vị trí ô chứa kết quả tối ưu vừa ghi nhớ, ta truy ngược lại về điểm bắt đầu của nghiệm: đó có thể là những ô đầu tiên trong bảng phương án (tức là vị trí chứa nghiệm của bài toán cơ sở), hay đó cũng có thể là ô của bảng phương án đạt giá trị ban đầu.

III) Bài toán

1. Bài toán dãy con tăng dài nhất

Đề bài: Tìm dãy con dài nhất của một dãy đã cho. Các phần tử có thể không liên tiếp nhau.

Ví dụ dãy: 1 2 5 4 6 3 8 9 7 ⇒ Kết quả dãy con tăng dài nhất là: 1 2 5 6 8 9

a. Phân tích:

Đầu vào input là một cặp (a,n), trong đó a là chuỗi đầu vào, n là độ dài chuỗi đó.

Output là 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, bằng độ dài dãy con dài nhất đến j cộng 1 khi ghép thêm a_i vào sau, với j<i, và $a_j < a_i$.

Công thức truy hồi: $L(i) = \max\{L(j)\} + 1$ với $j<i$, $a_j < a_i$.

Cơ sở quy hoạch động: $L(0) = 0$; $L(1) = 1$.

b. Bảng phương án:

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

Dựa vào bảng phương án:

Giá trị tối ưu: $\max (L(i))_{i = 1...n} = 6$

c. Bảng phương án V2:

Để đơn giản hơn trong việc lập bảng phương án, chúng ta có thể sử dụng một cách khác như sau:

B1: Tạo bảng chứa dãy L(i)

- Có độ dài bằng độ dài a(i)
- Giá trị các phần tử bằng 1

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

B2: Sử dụng thuật toán sau để cập nhật giá trị cho L(i)

```
for i in range(0, n):
    for j in range(0, i):
        if(a[i] > a[j] and result[i] < result[j] + 1):
            result[i] +=1
```

d. Truy vết tìm nghiệm:

Bắt đầu từ vị trí L(i) đạt max

Tại mỗi bước lùi về trước aj, L(j), mà aj < ai và L(j) = L(i) - 1

Kết thúc khi đến đầu dãy □ L(j) = 0

Note: có thể có nhiều giá trị cùng thoả mãn => có thể chọn 1 hoặc tất cả

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

e. Mã nguồn giải bài toán bằng phương pháp QHĐ

Lập bảng phương án và tìm số phần tử dãy con dài nhất:

```
# find Longest Increasing Subsequence
def maxium_value_ls(a, n):
    # create result table
    result = [1] * n
    for i in range(0, n):
        for j in range(0, i):
            # update result table
            if (a[i] > a[j] and result[i] < result[j] + 1):
                result[i] += 1
    return max(result), result
```

Truy vết và tìm nghiệm:

```
# solution of LIS
def findSolution(max, result, a):
    position = 0
    pos_solution = []
    solution = []
    for i in range(0, len(result)):
        if (result[i] == max):
            position = i
```

```
    pos_solution.append(position)
    solution.append(a[position])
    for i in range(position, 0, -1):
        if (result[position] == (result[i - 1] + 1)):
            position = i - 1
            pos_solution.append(i - 1)
            solution.append(a[position])
    return solution
```

Định dạng lại nghiệm:

```
def formatSolution(solution):
    result = []
    for i in range(0, len(solution)):
        result.append(solution.pop())
    return result
```

f. Mã nguồn giải bài toán bằng phương pháp đệ quy

Sử dụng công thức truy hồi:

$$ans = \max(ans, 1 + LIS\_ending\_number(a, i))$$

ta có Source code minh hoạ bằng ngôn ngữ Python như sau:

```
def LongestIncreasingSubsequence(a):
    LIS = 1
    for i in range(0, len(a)):
        LIS = max(LIS, LIS_ending_number(a, i))
    return LIS
```

```
def LIS_ending_number(a, current):
    if (a == 0):
        return 1
    ans = 1
    for i in range(current - 1, -1, -1):
        if (a[i] < a[current]):
            ans = max(ans, 1 + LIS_ending_number(a, i))
    return ans

# test case
a = [1, 2, 5, 4, 6, 3, 8, 9, 7]
print("input: ", a)
print("Longest Increasing Subsequence is ", LongestIncreasingSubsequence(a))
```

g. So sánh giữa đệ quy với quy hoạch động

Giải bài toán bằng phương pháp quy hoạch động

- Chúng ta có một mảng lưu trữ lại giá trị các bài toán con đã giải (result table)
- Việc truy vết tìm nghiệm với phương pháp QHĐ đơn giản

Giải bài toán bằng phương pháp đệ quy

- Chúng ta không có mảng lưu lại giá trị các bài toán con đã giải
- Việc truy vết tìm nghiệm với phương pháp đệ quy không đơn giản

⇒ Cần cải tiến hơn trong khi giải bằng phương pháp đệ quy để có thể truy vết tìm nghiệm

Độ phức tạp của thuật toán:

Thời gian: **Quy hoạch động = Đệ quy = O(nlog(n)).** Không gian: **Quy hoạch động** tốn **thêm một mảng lưu trữ** các giá trị bài toán con = L[i] (i = (0,n) trong đó n là độ dài của mảng input) đã giải được so với giải bằng phương pháp **đệ quy.**

2. Bài toán hệ số nhị thức (Binomial Coefficient)

a. Đề bài:

Một hệ số nhị thức C(n,k) có thể được định nghĩa là hệ số x^k trong sự mở rộng của (1+x)^n

Ví dụ: (a+b)^2 = a^2 + 2ab + b^2 ➔ hệ số nhị thức là 1 2 1

// trong đó, n = 2

// C(2, 1) = 2

b. Phân tích:

Input : n là bậc của biểu thức,

k vị trí của phần tử ta cần tính hệ số trong biểu thức //C(n,k)

Output: đưa ra giá trị của C(n,k)

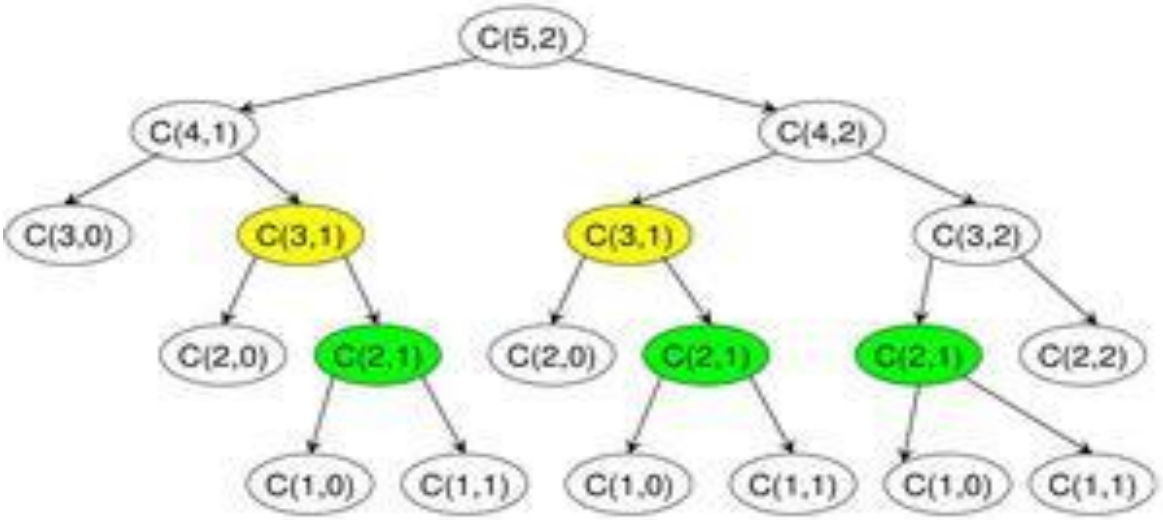
c. Xử lý bằng phương pháp đệ quy:

Khi xử dụng phương pháp đệ quy, chúng ta sẽ có công thức truy hồi như sau:

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

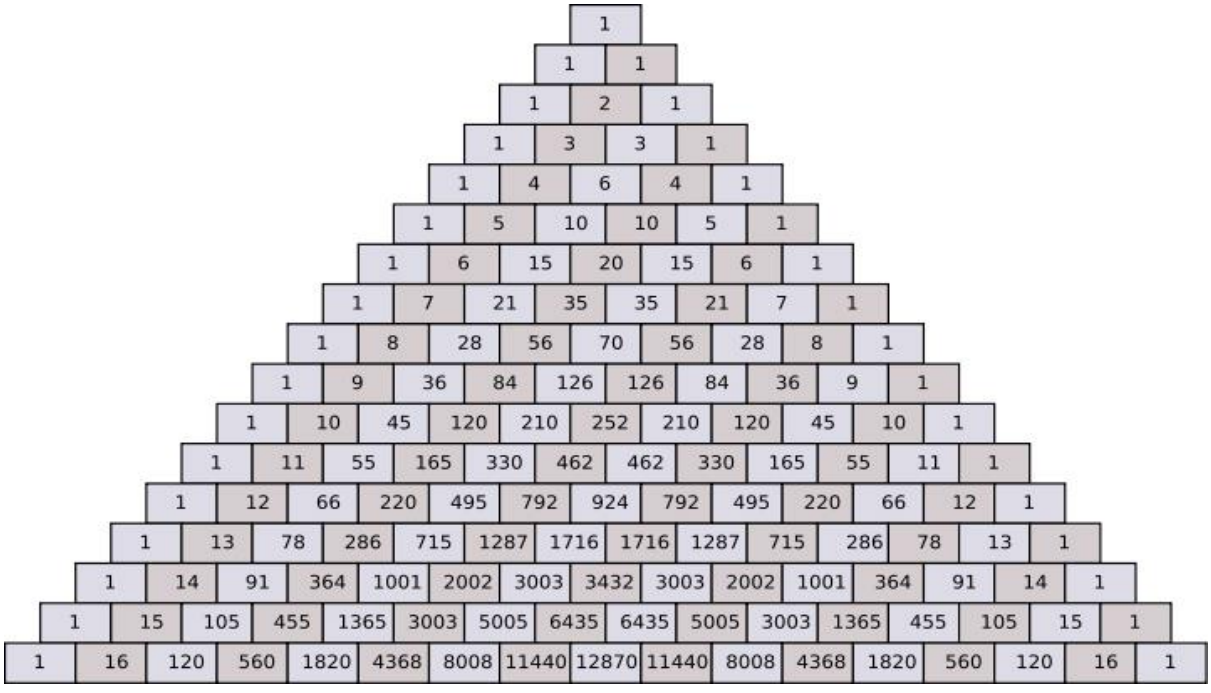
$$C(n, 0) = C(n, n) = 1$$

Với công thức truy hồi như vậy ta có thể biểu diễn cách hoạt động bằng một cây như sau:



⇨ Ta còn có thể biểu diễn bằng một tam giác pascal

Ví dụ: với n = 4 và k = 2



Áp dụng phương pháp đệ quy thì chúng ta sẽ phải giải đi giải lại các bài toán con mặc dù chúng ta đã giải ra kết quả ở trước đó.

⇨ Chỗ vướng mắc này chúng ta có thể chuyển sang sử dụng phương pháp quy hoạch động để khắc phục.

d. Giải bài toán bằng phương pháp đệ quy:

Công thức truy hồi:

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

$$C(n, 0) = C(n, n) = 1$$

Áp dụng công thức truy hồi ở trên chúng ta có thể xây dựng source code giải bài toán bằng phương pháp đệ như sau:

```
def binomialCoeff(n, k):
    if (k == 0 or k == n):
        return 1
    elif (k > n):
        return 0
    else:
        return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k)

print(binomialCoeff(5, 2))
```

e. Giải bài toán bằng phương pháp quy hoạch động:

```
def binomialCoefDP(n, k):
    # create result table
    C = []
    for i in range(0, n + 1):
        C.append([])
        for j in range(0, k + 1):
            C[i].append(0)
    # create result table
    # C = [[0 for x in range(k + 1)] for x in range(n + 1)]
    for i in range(n + 1):
        for j in range(min(i, k) + 1):
            # Base Cases
            if j == 0 or j == i:
                C[i][j] = 1
            else:
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j]
    return C[n][k], C
number, C = binomialCoefDP(5, 3)
print(C)
print(number)
```

f. So sánh giải bài toán bằng phương pháp đệ quy với QHĐ

Với phương pháp đệ quy, chúng ta có thể dễ dàng nhìn thấy phần được tô màu trên cách biểu diễn bằng cây là các bài toán con trùng nhau

➔ Chúng ta đã giải bài toán con trước đó, nhưng vẫn phải giải lại

➔ Gây ra tăng độ phức tạp của thuật toán

Với phương pháp QHĐ, chúng ta đã khắc phục nhược điểm trên bằng một mảng lưu trữ các phần tử

➔ Khắc phục được khó khăn của giải thuật đệ quy

➔ Chi phí thêm một mảng gồm n, k phần tử (n là bậc của nhị thức, k là vị trí hệ số cần tính)

3. LCS và Bài toán tối ưu không gian của LCS

a. Đề bài:

Từ bài toán LCS – Longest comment subsequence – bài toán tìm dãy con chung dài nhất (các phần tử có thể không liên tiếp nhau), hãy tối ưu hoá không gian của LCS.

b. Phân tích:

Input: cho 2 dãy đầu vào X, Y có độ dài lần lượt là m và n.

Output: tìm L là độ dài LCS của 2 dãy đã cho. Từ đó, tìm cách tối ưu hoá không gian của LCS.

c. Ý tưởng giải bài toán bằng vét cạn và đệ quy:

Để tìm ra độ phức tạp của phương pháp tiếp cận brute force, trước tiên chúng ta cần biết một chuỗi có độ dài n, sau đó ta cần tìm số lượng các chuỗi con khác nhau có thể có của chuỗi, tức là tìm số lượng các chuỗi con có độ dài từ 1 đến n-1 mà các phần tử không nhất thiết liền kề nhau, chỉ cần theo đúng thứ tự trước sau của chuỗi. Để liệt kê ra hết các các trường hợp, ta cần sử dụng kiến thức về hoán vị và tổ hợp để tính toán. Ví dụ số tổ hợp có 1 phần tử từ chuỗi n phần tử là C_n^1, 2 phần tử là C_n^2,… Tổng hợp lại ta có $\sum C_n^i = 2^n - 1$ ($với\ i = \overline{1,n}$) dãy con có thể có khác nhau không kể dãy có độ dài bằng 0. Điều này có nghĩa là độ phức tạp về thời gian của phương pháp brute force là $O(n \times 2^n)$, lưu ý thêm rằng là phải mất $O(n)$ thời gian để kiểm tra một dãy con có chung cho cả 2 chuỗi hay không. Với thời gian phức tạp này ta có thể nghĩ ngay đến QHĐ.

Giải pháp đơn giản cho vấn đề này là tạo ra tất cả các dãy con của cả 2 dãy đã cho và tìm dãy con phù hợp dài nhất. Giải pháp này là cấp số nhân về độ phức tạp thời gian. Chúng ta sẽ xem bài toán có sở hữu dấu hiệu nhận biết để lập trình theo hướng QHĐ hay không.

Trước hết về *Cấu trúc con tối ưu*:

Cho dãy đầu vào $X_{0\ldots m-1}$ và $Y_{0\ldots n-1}$ có độ dài tương ứng là m và n. Gọi $\mathrm{L}(X_{0\ldots m-1}, Y_{0\ldots n-1})$ là độ dài LCS của 2 dãy X và Y. Ta có công thức đệ quy sau:

$$\mathrm{L}(X_{0\ldots m-1}, Y_{0\ldots n-1}) = \begin{cases} 1 + \mathrm{L}(X_{0\ldots m-2}, Y_{0\ldots n-2}), với\ X_{m-1} = Y_{n-1} \\ \max(\mathrm{L}(X_{0\ldots m-2}, Y_{0\ldots n-1}), \mathrm{L}(X_{0\ldots m}, Y_{0\ldots n-2})), với\ X_m \neq Y_n \end{cases}$$

Ví dụ: Chuỗi đầu vào X = “AGGTAB” và Y = “GXTXAYB”. Nhận thấy các ký tự cuối cùng của X và Y khớp nhau, vậy độ dài của LCS có thể được viết: L(“AGGTAB”, “GXTXAYB”) = 1 + L (“AGGTA”, “GXTXAY”). Nhưng với “AGGTA”, “GXTXAY” thì ký tự cuối là khác nhau nên độ dài LCS của chúng là max(L(“AGGT”, “GXTXAY”), L(“AGGTA”, “GXTXA”)).

Qua trên ta thấy bài toán LCS có tính cấu trúc con tối ưu vì bài toán có thể được giải bằng cách sử dụng các giải pháp cho các bài toán con.

Về *Bài toán con chồng nhau*:

Với thuật toán đệ quy tương tự như trên, ta có cây đệ quy một phần như sau:

```
                        lcs("AXYT", "AYZX")
                       /
          lcs("AXY", "AYZX")            lcs("AXYT", "AYZ")
          /                              /
lcs("AX", "AYZX") lcs("AXY", "AYZ")   lcs("AXY", "AYZ") lcs("AXYT", "AY")
```

Trong cây đệ quy từng phần như trên, lcs(“AXY”, “AYZ”) được giải 2 lần. Nếu cây đệ quy được vẽ mở rộng hoàn chỉnh, thì chúng ta có thể thấy rằng có rất nhiều bài toán con được giải đi giải lại nhiều lần. Vì vậy, bài toán này có thuộc tính của bài toán con trùng nhau.

Mã giả mô phỏng thuật toán:

$LCS(X, Y, m, n) \equiv$

$if\ m = 0\ or\ n = 0$

$return\ 0$

$elif\ X_m = Y_{n:}$

$return\ 1 + LCS(X, Y, m, n-1)$

$else$:

$return$ max $(LCS(X, Y, m, n-1), LCS(X, Y, m-1, n))$

$end.$

Độ phức tạp thời gian theo cách tiếp cận đệ quy là $O(2^n)$ trong trường hợp xấu nhất và xấu nhất xảy ra khi tất cả các ký tự của X và Y là không trùng nhau, tức là LCS(X, Y) = 0.

Tóm lại, gọi n là độ dài của chuỗi X hoặc Y, vậy để giải bài toán LCS giữa hai chuỗi X và Y bằng phương pháp vét cạn thì độ phức tạp thời gian là $O(n \times 2^n)$ là rất lớn, và với đệ quy, độ phức tạp thời gian giảm còn $O(2^n)$ nhưng vẫn còn rất lớn. Không những thế, qua việc phân tích từ đệ quy trên thấy, bài toán LCS có chứa đặc điểm của bài toán giải được bằng QHĐ. Do đó ta có thể tránh được việc tính toán lại các bài toán con giống nhau bằng cách sử dụng bảng ghi nhớ.

d. Bài toán xử lý bằng phương pháp quy hoạch động:

Hàm mục tiêu: giả sử L(i, j) trả về độ dài dãy con chung lớn nhất giữa phần tử thứ $i-1$ của X và thứ $j-1$ của Y.

Công thức truy hồi:

$$\mathrm{L(i,j)} = \begin{cases} 1 + \mathrm{L}(i-1, j-1), \mathrm{X[i-1]} = \mathrm{Y[j-1]} \\ \mathrm{MAX}\big(\mathrm{L(i-1,j)}, \mathrm{L}(i, j-1)\big), \mathrm{X[i-1]} \neq \mathrm{Y[j-1]} \end{cases}$$

Cơ sở QHĐ:

$$\mathrm{L}(0, j) = \mathrm{L}(i, 0) = 0;\ \forall i = \overline{0, n}\ \ j = \overline{0, m}$$

Bảng phương án: ta có bảng 2 chiều $L_{(m+1)\times(n+1)}$ lưu trữ các giá trị về độ dài dãy con chung lớn nhất giữa X và Y. Giả sử ta có bảng phương án như sau:

L(i, j)	0	…	i – 1	i	…	n
0	0	0	0	0	0	0
⋮	…	…	…	…	…	…
j – 1	0	…	.	**a**	…	….
j	0	…	**b**	$\begin{cases} \boldsymbol{b+1, X[i] = Y[j]} \\ \boldsymbol{\max(a,b), X[i] \neq Y[j]} \end{cases}$	…	…
⋮	…	…	…	…	…	…
m	0	…	…	…	…	**result**

Giá trị tối ưu: khi cứ tiếp tục điền bảng phương án theo công thức như trên, kết quả tối ưu ta thu được nằm ở vị trí cuối cùng của bảng, tức là kết quả tối ưu ta cần tìm là $L(m, n)$.

Truy vết tìm nghiệm: tạo 1 mảng để chứa chuỗi LCS có độ dài bằng giá trị tại ô L(m, n) + 1. Ta bắt đầu duyệt từ vị trí tối ưu của bảng phương án – vị trí mà $L(i, j)$ đạt max, vị trí L(m, n). Tại mỗi bước lùi về trước:

Nếu ký tự tại $X[i-1] \equiv Y[j-1]$ thì ký tự đó nằm trong chuỗi con chung, ta tiếp lục lặp với $i = i-1, j = j-1$.

Ngược lại, ta sẽ so sánh giá trị của $L(i-1, j)$ và $L(i, j-1)$ rồi đi theo hướng có giá trị lớn hơn, tức là tiến về trước 1 cột hoặc 1 hàng theo ô có giá trị lớn nhất.

Đánh giá: Qua việc phân tích trên có thể thấy, độ phức tạp về thời gian của bài toán LCS khi sử dụng thuật toán QHĐ đã được cải tiến, giảm xuống còn là $O(m \times n)$.

e. Mở rộng – tìm hiểu giải pháp tối ưu hoá không gian của LCS:

Bằng phương pháp QHĐ, với m, n là độ dài 2 xâu bất kỳ X và Y, ta có một bảng tra lưu trữ kết quả để truy vết thông tin tìm nghiệm. Dễ dàng thấy bảng đó có kích

thước là $m \times n$, tương đương với việc độ phức tạp về không gian của bài toán cũng là $O((m+1) \times (n+1))$. Với sự phát triển nhanh chóng về kỹ thuật hiện nay thì việc lưu trữ một không gian từ khá đến rất lớn của một hay nhiều bài toán không là một vấn đề lớn. Tuy nhiên, nếu có thể xử lý bài toán vừa nhanh vừa tiết kiệm không gian thì lại càng tốt. Ta có một giải pháp tối ưu hoá không gian của LCS (Space optimized solution LCS – SOS LCS).

Nhận thấy với bài toán LCS, với mỗi lần lặp của vòng ngoài thì ta chỉ cần các giá trị từ cột của hàng trước đó. Vì vậy, việc lưu trữ tất cả như cũ là không cần thiết, ta chỉ cần lưu trữ 2 hàng và sử dụng chúng thay luân phiên nhau. Theo cách đó, không gian sử dụng sẽ giảm từ $O((m+1) \times (n+1))$ còn $O(2 \times (n+1))$.

Trước hết ta khởi tạo một bảng cỡ $2 \times (n+1)$ như sau:

L	0	1	…	i	…	n
0	0	0	…	0	0	0
1	0	0	…	0	0	0

Sau đó, ta thực hiện phép toán với công thức truy hồi LCS như trên, với mỗi vòng lặp, ta có:

Với L(1, j):

L	0	…	j-1	j	…	n
0	**0**	**.**	**.**	**b**	**…**	**.**
1	**0**	.	**a**	$\begin{bmatrix} a+1, X[i] = Y[j-1] \\ \max(a,0), X[i] \neq Y[j-1] \end{bmatrix}$	…	.

Hoặc L(0, j):

L	0	…	j-1	j	…	n
0	**0**	.	**a**	$\begin{bmatrix} a+1, X[i] = Y[j-1] \\ \max(a,c), X[i] \neq Y[j-1] \end{bmatrix}$	…	.
1	**0**	…	.	**c**	…	.

Nhận xét:

- Với bài toán SOS LCS: kết quả cần tìm nằm ở vị trí $L(bi, n)$ với $\begin{cases} bi = 0, \text{nếu m chẵn} \\ bi = 1, \text{nếu m lẻ} \end{cases}$.
- Độ phức tạp thời gian của bài toán vẫn là $O(m \times n)$, nhưng độ phức tạp về không gian đã được tối ưu hoá còn $O(m)$ hoặc $O(n)$ và tối ưu nhất là $O(\min(m, n))$.

IV) Tài liệu tham khảo

Slide bài giảng của cô Nguyễn Thị Hồng Minh

https://www.geeksforgeeks.org/dynamic-programming

Quy hoạch động – Wikipedia tiếng Việt

Thuật toán Quy hoạch động - Một thuật toán thần thánh | TopDev