**Chuyên đề hội thảo khối chuyên Tin**

**Chủ đề:** Quy hoạch động

*I.Lý thuyết*

1.Khái niệm

**- Quy hoạch động** (Dynamic Programming) là một phương pháp giảm thời gian chạy các của các bài toán thể hiện tính chất của **bài toán con gối nhau** và **cấu trúc con tối ưu**.

- **Cấu trúc con tối ưu**: có nghĩa là các lời giải tối ưu của bài toán con nhỏ hơn có thể dùng để giải các bài toán toàn cục.

- **Bài toán con gối nhau**: các bài toán lớn hơn sẽ được chia thành các bài toán con nhỏ hơn, và sẽ chia đến khi bài toán đơn giản dễ tìm lời giải

-> Trong các bài toán trên, phương pháp **quy hoạch động** sẽ được dùng để tránh thực hiện một bài toán nhiều lần bằng cách lư kết quả của bài toán để sử dụng trong bài toán sau lớn hơn.

- Hiểu một cách đơn giản hơn: Quy hoạch động sẽ lưu những kết quả của bài toán trước nhỏ hơn để giải bài toán lớn hơn.

2.Cách tiếp cận

Quy hoạch động thường dùng một trong hai cách tiếp cận:

* **top-down**(*Từ trên xuống*): Bài toán được chia thành các bài toán con, các bài toán con này được giải và lời giải được ghi nhớ để phòng trường hợp cần dùng lại chúng. Đây là đệ quy và lưu trữ được kết hợp với nhau.
* [**bottom-up**](https://vi.wikipedia.org/w/index.php?title=Bottom-up&action=edit&redlink=1)(*Từ dưới lên*): Tất cả các bài toán con có thể cần đến đều được giải trước, sau đó được dùng để xây dựng lời giải cho các bài toán lớn hơn. Cách tiếp cận này hơi tốt hơn về không gian bộ nhớ dùng cho ngăn xếp và số lời gọi hàm. Tuy nhiên, đôi khi việc xác định tất cả các bài toán con cần thiết cho việc giải quyết bài toán cho trước không được trực giác.

3.Phương pháp giải bài bằng quy hoạch động.

- Tìm cách biểu diễn kết quả của bài toán tổng quát và bài toán con.

- Tìm công thức đệ quy biểu diễn kết quả của bài toán lớn thông qua kết quả của bài toán con (tức tìm quan hệ giữa bài toán lớn với bài toán con).

- Khởi tạo kết quả của các bài toán nhỏ nhất (các bài toán đơn giản dễ tìm lời giải)

- Dựa vào cách tổ chức dữ liệu để truy vết kết quả tối ưu.

*II.Bài tập áp dụng*

1.Một số bài toán kinh điển

**\* Dãy con tăng đơn điệu dài nhất** (Lowest Increasing Subsequence)

a) Mô hình bài toán:

Cho dãy A gồm n phần tử: a1, a2, a3, …, an.

Hãy tìm một dãy con tăng (dãy con gồm các phần tử tăng dần) có nhiều phần tử nhất của dãy (Các phần tử trong dãy vẫn giữ nguyên thứ tự).

b) Phân tích bài toán:

Do các phần tử trong dãy vẫn giữ nguyên thứ tự so với mảng gốc

* Nếu phần tử ai (1 i n) xuất hiện trong dãy con tăng thì nó sẽ luôn xuất hiện sau những phần tử nằm trước nó.

Nhận xét: Giả sử ai là phần tử cuối cùng của một dãy con tăng thì đoạn con trước phần tử ai  cũng là dãy con tăng với độ dài nhỏ hơn 1 và các phần tử trong dãy con sẽ luôn nhỏ hơn ai.

c) Công thức Quy hoạch động

Gọi len[i]: độ dài dãy con tăng dài nhất với phần tử kết thúc là phần tử ai.

Ta sẽ có công thức sau:

* Len[0] = 0 (dãy gồm 0 phần tử chỉ có dãy con có độ dài 0)
* Len[i] = max(len[j]) + 1 với j là các phần tử thoả mãn ai > aj. (0 j < i)

(bổ sung thêm phần tử ai vào cuối dãy con tăng dài nhất với kết thúc < ai để tạo ra một dãy con tăng mới dài nhất kết thúc ở ai)

Ta sẽ có đoạn chương trình sau:

**for** i = 1 -> n:

len[i] = 1; // = len[0] + 1

**for** j = 1 -> j – 1:

**if** (a[j] < a[i]):

len[i] = **max**(len[i], len[j] + 1)

Độ phức tạp thuật toán:

* Bộ nhớ: O(n)
* Thời gian: O(n2)

**\* Dãy con chung dài nhất** (Longest Common Subsequence)

a) Mô hình bài toán:

Cho 2 dãy A và B:

* Dãy A gồm n phần tử: a1, a2, a3, …, an.
* Dãy B gồm m phần tử: b1, b2, b3, …, bm.

Hãy tìm một dãy con (không nhất thiết phải liên tiếp) chung của 2 dãy A và B sao cho độ dài dãy con là dài nhất

b) Phân tích bài toán:

Tương tự như bài toán trên, do các phần tử trong dãy vẫn giữ nguyên thứ tự so với mảng gốc

* Nếu phần tử ai (1 i n) xuất hiện trong dãy con chung thì nó sẽ luôn xuất hiện sau những phần tử nằm trước nó.

c) Công thức Quy hoạch động:

Gọi len[i, j]: Độ dài dãy con chung dài nhất của 2 dãy a1, a2, a3, …, ai và b1, b2, b3, …, bm.

Ta có công thức sau:

* len[i, 0] = 0 (0 i n) (Dãy con chung duy nhất là dãy rỗng)
* len[0, j] = 0 (0 j m) (Dãy con chung duy nhất là dãy rỗng)
* len[i, j] = len[i-1, j-1] + 1 nếu ai = bj (thêm kí tự ai/ bj vào xâu con chúng)
* len[i, j] = max(len[i-1, j], len[i, j-1]) nếu ai  bj (dãy con chung của tập nhỏ hơn sẽ luôn là dãy con chung của tập lớn hơn)

Ta sẽ có đoạn chương trình sau:

**for** j = 1 -> m:

len[0, j] = 0

**for** i = 1 -> n:

len[i, 0] = 0

**for** j = 1 -> m:

**if (**a[i] = b[j]):

len[i, j] = len[i - 1, j - 1] + 1

**else:**

len[i, j] = **max**(len[i - 1, j] , len[i, j - 1])

Độ phức tạp thuật toán:

* Bộ nhớ: O(n m)
* Thời gian: O(n m)

**\*Bài toán cái túi** (Knapsack problem)

a) Mô hình bài toán:

Cho *n* đồ vật khác nhau, đồ vật thứ *i* có trọng lượng *Wi*​ và giá trị *Vi*​. Bạn mang theo một chiếc túi có tải trọng tối đa là *W.* Nhiệm vụ của bạn là chọn ra các đồ vật để cho vào túi sao cho tổng giá trị của các đồ vật lấy được là lớn nhất có thể?

b) Phân tích bài toán:

Do đây là một bài toán chọn đồ nên mỗi khi ta xét một đồ vật thì ta phải lựa xem có nên chọn bỏ vào túi hay không?

Nhận thấy rằng bài này có thể giải được bằng cách tạo các xâu nhị phân độ dài *n.* Nếu vị trí thứ *i* của xâu nhị phân trên là 1 thì ta sẽ cho vật thứ *i* vào túi, bỏ qua nếu vị trí thứ *i* của xâu nhị phân trên là 0. Tuy nhiên, làm vậy sẽ ảnh hưởng nhiều đến độ phức tạp thời gian nên ta có công thức quy hoạch động sau để giải bài toán.

c) Công thức quy hoạch động:

Gọi *dp[i][j]* là tổng giá trị lớn nhất của các đồ vật lấy được khi xét các đồ vật từ *1* tới *i* và giới hạn tổng trọng lượng của chúng là *j*. Kết quả của bài toán là tổng giá trị lớn nhất chọn được trong *n* vật với giới hạn trọng lượng là *W*, nói cách khác thì chính là *dp*[*n*][*W*].

Như đã nói ở trên, khi xét vật i thì ta có 2 phương án để lựa chọn:

● Nếu vật thứ *i* được chọn vào phương án tối ưu, thì tải trọng còn lại có thể sử dụng là (*j − Wi​*) cho *(i−1)* đồ vật phía trước, và ta được thêm giá trị *Vi*​ của vật thứ *i* (Trường hợp này chỉ xảy ra khi *j* ≥ *Wi )*.

● Nếu vật thứ *i* không được chọn vào phương án tối ưu, thì kết quả tối ưu sẽ được chọn trong *(i − 1)* đồ vật trước đó với giới hạn trọng lượng vẫn là *j.*

Từ đó, ta rút ra được công thức quy hoạch động sau:

For i = 1 -> n:

For j = 1 -> W:

If j >= W:

dp[i, j] = max(dp[i – 1, j], dp[i – 1][j – w[i]] + v[i]

Else:

dp[i, j] = dp[i – 1, j]

Ta có đoạn chương trình con sau:

Độ phức tạp thuật toán:

- Thời gian: O(n x W)

- Bộ nhớ: O(n x W)

## \* Khoảng cách chỉnh sửa (Edit Distance)

a) Mô hình bài toán:

*Khoảng cách chỉnh sửa* giữa hai xâu là số lượng thao tác tối thiểu cần thiết để chuyển đổi một xâu thành xâu kia.

Các thao tác được phép là:

* Thêm một ký tự vào xâu.
* Xóa một ký tự khỏi xâu.
* Thay thế một ký tự trong xâu.

Ví dụ: khoảng cách chỉnh sửa giữa LOVE và MOVIE là 2, vì trước tiên có thể thay thế L bằng M, sau đó thêm I.

Nhiệm vụ là tính toán khoảng cách chỉnh sửa giữa hai xâu.

Input:

* Dòng đầu vào đầu tiên có một xâu chứa **n** ký tự giữa A–Z.
* Dòng đầu vào thứ hai có một xâu chứa các ký tự **m** giữa A–Z

Output:

* In một số nguyên: khoảng cách chỉnh sửa giữa các xâu.

b) Phân tích bài toán:

Do chỉ có ba thao tác để thay thế xâu:

Nên ta có 3 trường hợp:

* Thêm một ký tự vào xâu.
* Xóa một ký tự khỏi xâu.
* Thay thế một ký tự trong xâu.

Example:

Để tạo ra xâu : “Love”.

Thì có 3 cách là:

+) “Lov”+”e”=”Love”

+) “Lovie”-“i”=”Love”(hoặc các kí tự khác thừa mà sau khi xóa đi ta đc xâu “love”)

+)”Lovy” –“y”+”e”=”Love”(hoặc các kí tự khác mà sau khi thay thế kí tự đấy ta được xâu “Love”)

c) Công thức Quy hoạch động

Gọi f[i][j]: khoảng cách chỉnh sủa từ i kí tự đầu tiên của xâu thứ nhất sang j kí tự đầu tiên của xâu thứ 2.

Gọi xâu đầu tiên là s,xâu thứ 2 là s1.

Khởi tạo :

F[i][0]=i;(i=1->s.size)

F[0][i]=i;(i=1->s1.size)

Ta sẽ có công thức sau:

* f[i][j]**=**f[i-1][j-1] ( nếu s(i) giống với s1(j) )

f[i][j]=min(f[i][j-1]**+**1,f[i-1][j]**+**1,f[i-1][j-1]**+**1)

(với những trường hợp còn lại)

Ta có đoạn chương trình sau:

**for** i= 1 -> s1.**size**():

F[0, i] = i

**for** i= 1 -> s.**size**():

F[i, 0] = i

**for** j= 1 -> s1.**size**():

**if**(s[i] = s1[j]):

F[i, j] = F[i - 1, j - 1]

**else:**

F[i, j] = **min**(F[i - 1, j], F[i, j - 1], F[i - 1, j - 1]) + 1

Độ phức tạp thuật toán:

* Bộ nhớ: O(n \* m)
* Thời gian: O(n\*m)

2.Một số bài tập khác.

### \*Trò Chơi Negiko ([Bedao Regular Contest 08 - NEGIKO](https://oj.vnoi.info/problem/bedao_r08_negiko))

a)Mô hình bài toán

Nam là một đại kỳ thủ trong trò chơi NEGIKO, với một biệt danh cực kì nổi tiếng là "Bố của các kỳ thủ NEGIKO". Anh ấy giỏi chơi trò này đến nỗi anh có thể nhìn thấy được có bao nhiêu cách để giành chiến thắng. Tuy nhiên, vì một ngày nọ anh ấy thức khuya leo rank thách đấu nên ngày hôm đó anh ấy không thể đếm được chính xác có bao nhiêu cách. Vì vậy nên bạn, với tư cách là bạn thân chí cốt của anh ấy, hãy tạo ra một chương trình giúp đếm số cách để Nam chiến thắng.

Biết trò chơi NEGIKO có luật chơi như sau:

Neko cầm n thẻ bài, mỗi thẻ bài có giá trị a[i].

Đối thủ cầm m thẻ bài, mỗi thẻ bài có giá trị b[i].

Mỗi ván chơi Nam sẽ chọn bất kì k thẻ bài của anh ấy và đối thủ sẽ chọn bất kì k thẻ bài của cậu ấy.Nam sẽ giành chiến thắng khi trong k thẻ bài từ Nam và k thẻ bài từ đối thủ được chọn:

* Thẻ bài có giá trị cao nhất của Nam lớn hơn thẻ bài có giá trị cao nhất của đối thủ
* Thẻ bài có giá trị cao thứ hai của Nam lớn hơn thẻ bài có giá trị cao thứ hai của đối thủ, …
* Thẻ bài có giá trị cao thứ k của Nam lớn hơn thẻ bài có giá trị cao thứ k của đối thủ.

Yêu cầu: Hãy đếm số cách chọn để Nam giành chiến thắng.

Dòng đầu nhập vào 3 số nguyên n,m,k

Dòng thứ hai chứa n số nguyên dương a1, a2, …,an

Dòng thứ ba chứa m số nguyên dương b1,b2, …,bn

b) Ý tưởng thuật toán

Ý tưởng là sort 2 dãy a,b tăng dần xong chọn lần lượt các thẻ bài (i,j) để ai>bj. Khi mà xét đến dpi,j,k ta có 3 lựa chọn ở trạng thái trước là không chọn i, không chọn j và chọn i, j.

* Trường hợp không chọn i: dp[i, j, k] += dp [i−1, j, k].
* Trường hợp không chọn j: dp[i, j, k] +=dp [i, j-1−1, k].
* Trường hợp chọn i và j, chỉ xảy ra khi ai>bj:dp[i, j, k]+=dp[i−1, j−1, k−1].

Nhưng ta nhận thấy là trường hợp không chọn i và trường hợp không chọn j nó có phần trùng nhau, đó là các cách chọn mà không chọn cả i và j, khi đó các cách chọn này bị tính 2 lần, số cách chọn không chọn cả i và j là dp[i−1, j−1, k] nên ta cần:  dp[i, j, k] = dp[i-1, j-1, k].

Nguồn: <https://oj.vnoi.info/problem/bedao_r08_negiko/editorial>

## **\*Tổng chữ số**([**Atcoder Educational DP Contest S - Digit Sum**](https://atcoder.jp/contests/dp/tasks/dp_s)**)**

**Đề bài**:Hãy đếm số lượng số nguyên (modulo 109+7) trong phạm vi từ 1 đến **K** (tính cả 1 và **K**) thoả mãn:

* Tổng các chữ số trong biểu diễn thập phân của số đó là bội của **D**.

**Lời giải**: (nguồn tại [đây](https://oj.vnoi.info/problem/atcoder_dp_s/editorial))

Ta có thể dùng [quy hoạch động chữ số](https://codeforces.cc/blog/entry/53960) để giải bài toán này. Gọi dp[i][j] là số cách để xây dựng một số có |K| chữ số và tổng chữ số chia hết D, sao cho i chữ số đầu cố định và tổng i chữ số đó chia D dư j.

Để tính đáp án, ta duyệt qua tất cả các tiền tố của xâu K và tính số cách để điền phần hậu tố còn lại của xâu bằng mảng quy hoạch động trên.

Công thức: dp[i][j] =

## **\*Ghép cặp**([**Atcoder Educational DP Contest O – Matching**](https://atcoder.jp/contests/dp/tasks/dp_o)**)**

**Đề bài**: Có N người đàn ông và N người phụ nữ đều được đánh số 1,2,...,N.

Với mỗi cặp số i,j (1≤i,j≤N), **độ hợp nhau** của người đàn ông i và người phụ nữ j là ai,j. Nếu ai,j =1 thì ông i và bà j hợp nhau, nếu ai,j =0 thì ngược lại.

Taro tìm cách chia N người đàn ông và N người phụ nữ thành N cặp, trong đó mỗi cặp gồm một người đàn ông và một người phụ nữ hợp nhau.

Hãy tìm số cách khác nhau mà Taro có thể chia, modulo 109+7

**Lời giải**: (nguồn tại [đây](https://oj.vnoi.info/problem/atcoder_dp_s/editorial))

Chúng ta định nghĩa dp[S] là số cách để ghép những người trong tập S vào |S| người đàn ông đầu tiên, từ đó ta có công thức như sau: dp[S]=∑dp[S/x]:a[|S|][x]=1

(Dấu : ở đây nghĩa là "thỏa mãn điều kiện")

(S/x nghĩa là tập S bỏ phần tử x)

Lưu ý: các tập S có thể được biểu diễn dưới dạng chuỗi nhị phân có 2N bit (bitmask).

Có thể giải thích công thức trên như sau: Số cách ghép cho tập S mà trong đó có người phụ nữ x nào đó sẽ là tổng tất cả các cách ghép không có người phụ nữ x và người phụ nữ x và người đàn ông |S| phải hợp nhau.

Base case của chúng ta là tập rỗng, có giá trị bằng 1 (tập rỗng có thể được tính là một cách ghép duy nhất có 0 cặp).

**\*Một số bài tập khác:**

[MAIL - Thư điện tử](http://ntucoder.net/Problem/Details/4504?fbclid=IwAR0bRwquqYtJkU3EKw5lb7HHnAqhVDitRUbZVdsc0YgWdQBTadaoE-lWU6Y)

[BALLGAME - Trò chơi với những viên bi](http://ntucoder.net/Problem/Details/5624?fbclid=IwAR1_B6yNRVRxFJAmemX92lYaVbKsp5kK1m-h73cvjy7mx0igO0JmApKjGmQ)

[DICE – Đổ xúc xắc](http://ntucoder.net/Problem/Details/4461?fbclid=IwAR2DQ965g_y-ix-ujaoNHHxKKfXLA_rE_V4DhcHzB5leXhcCFTcwTKI_dw0)

[Xếp hàng mua vé](https://oj.vnoi.info/problem/nktick?fbclid=IwAR178D_AE68SpWpbFmLUxNs-BoahHf0IUtLC3fqKjpL-zRH-7ZCOqsStuAA)

[Making tower](https://codeforces.com/problemset/problem/1706/B?fbclid=IwAR3S-cdJ9vwsH1obhAzo8etTJB5ROF9FucTnWoKfvc_J-4UigwJYleO3o0o)

[A Wide, Wide Graph](https://codeforces.com/problemset/problem/1805/D)

Extremely hard(siêu khó): [Lisa’s Sequences](https://codeforces.com/problemset/problem/1773/L)

*III.Một số phương pháp cải tiến thời gian chạy, bộ nhớ của thuật toán*

1.Đảo nhãn

**Đảo nhãn**: thay đổi các biến với nhau để tạo thuận lợi cho thời gian chạy và bộ nhớ trong một số trường hợp

Xét bài: Dãy con tăng dài nhất(LIS)

- Ta có F[i] = len: với len là độ dài dãy con tăng dài nhất kết thúc ở ai

- Thay vì đó, ta có thể định nghĩa F[len] = ai: với ai là phần tử nhỏ nhất có độ dài dãy con tăng kết thúc ở chính nó là len.

Kết hợp với tham lam:

* F[0] = -∞; F[len] = +∞ với mọi 0 < len;
* Ta có công thức quy hoạch động:

+ F[len] = ai nếu F[len – 1] < ai và ai ≤ F[len]

* Sau mỗi thao tác trên, các phần tử của F luôn có thứ tự tăng dần nên ta có thể sử dụng

\*Chứng minh công thức tham lam:

1 - F luôn tối ưu sau mỗi thao tác

* Với các thao tác trên, do ai ≤ F[len] nên F[len] sẽ luôn tối ưu sau mỗi bước(và chỉ có duy nhất một vị trí len trong F thoả mãn điều này do các phần tử trong F có thứ tự tăng dần)
* Như đã giải thích bên trên, để dãy con tăng độ dài kết thúc ở ai thì ai > phần tử kết thúc ở dãy con độ dài len – 1
* Do F[len] luôn nhỏ nhất có thể nên ai luôn kết thúc ở dãy con lớn nhất có thể

2 – Các phần tử trong mảng F luôn tăng dần sau mỗi thao tác

* Giả sử các phần tử trong mảng F đã tăng dần
* Trước mỗi thao tác, ta có F[len - 1] < F[len] với mọi len
* Sau thao tác:

F[len – 1] < ai và ai ≤ F[len] ⬄ F[len – 1] < F[len]

Ta có đoạn chương trình

**int lower\_bound**(**int** l, **int** r, **int** val)

{

kq = -1

**while**(l <= r):

mid = (l + r)/2

**if**(val <= F[mid]):

kq = mid

l = mid + 1

**else:**

r = mid - 1

return kq

}

F[0] = -∞

**for** i= 1 -> n:

F[mid] = +∞

**for** i= 1 -> n:

vt = **lower\_bound**(1, n, a[i])

F[vt] = a[i]

Độ phức tạp thuật toán:

* Bộ nhớ: O(n)
* Thời gian: O(n)

Một số bài tập thực hành:

* [VNOJ - BINPACK](https://oj.vnoi.info/problem/binpack/)
* [COMPUTER - VNOJ](https://oj.vnoi.info/problem/computer/)

**2.Chia để trị**2

**Chia để trị** là chia vấn đề ra thành nhiều phần nhỏ hơn

Bland joke: \*Chia để trị không đơn giản là thuật toán, đó là một hệ tư tưởng\*

**Xét bài:** Hai nhà máy gỗ(Two salmills)

Nguồn bài: [https://www.oi.edu.pl/old/ceoi2004](https://www.oi.edu.pl/old/ceoi2004/)

**Tóm tắt đề:**

Có **n** cây trên đồi rải rác từ đầu đến cuối đỉnh và cần đưa đến các nhà máy cưa

Mỗi cây có trọng lượng wi và khoảng cách từ nó đến cây tiếp theo là di (dn sẽ là khoảng cách từ cây n đến chân đồi)

Có một nhà máy cưa ở chân đồi. Tìm cách xây 2 nhà máy cưa ở trên đồi sao cho chi phí đem đi vận chuyển đến nhà máy cưa

Chi phí được tính bằng: wi \* leni với leni là khoảng cách từ cây thứ i đến nhà máy cưa gần nhất

**Lời giải**:(nguồn ở [đây](https://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong.md#2-chia-%C4%91%E1%BB%83-tr%E1%BB%8B))

*Nhận xét*: nhà máy sẽ luôn xây ở vị trí 1 cái cây, không xây ở giữa 2 cái cây do nếu xây ở giữa 2 cây, đường đi đến nhà máy của các cây đằng trước sẽ dài thêm và không có lợi ích gì trong việc rút ngắn đoạn đường các cây sau nhà máy

Trước hết ta sẽ giải quyết vấn đề tính chi phí vận chuyển nếu biết vị trí của hai nhà máy đặt thêm.

Nếu ta có thể tính được chi phí này trong O(1), bài toán sẽ có thể giải được trong O(N2) - ta có thể for hết các cặp vị trí có thể đặt nhà máy.

Gọi:

* **sumWi** là tổng của các **wj** với i≤j.
* **sumDi** là tổng của các **dj** với i≤j.
* **sumWSi** là tổng của các **wj**∗**sumDj** với i≤j.

Khi đó cost(L,R) là chi phí vận chuyển các cây có chỉ số trong đoạn [L,R] đến nhà máy đặt ở R là: sumWSL−sumWSR−sumDR∗(sumWL−sumWR).

Như vậy ta có thể xây dựng hàm eval(i,j) = chi phí nếu đặt thêm hai nhà máy ở i và j = cost(1,i)+cost(i+1,j)+cost(j+1,n+1)

Tuy nhiên lời giải O(N2) là chưa đủ tốt để có thể giải quyết trọn vẹn bài toán này.

Gọi best(i) là vị trí j>i tốt nhất nếu ta đã đặt một nhà máy ở i.

Như vậy kết quả của bài toán sẽ là min(eval(i,besti) với 1≤i<n.

Nhận xét:

* besti≤besti+1. Có thể viết tường minh công thức để chứng minh.
* Ta có thể tính các besti theo thứ tự bất kỳ. Vì các giá trị best không liên quan đến nhau nên VD ta có thể tính best(3) rồi best(1) và best(2).

Như vậy ta có thuật toán sử dụng tư tưởng chia để trị như sau:

Hàm solve(L,R,from,to) sẽ đi tính các best(L..R), biết rằng chúng nằm trong đoạn [from..to].

**void** **solve**(**int** L, **int** R, **int** from, **int** to) {

**if** (L **>** R) **return**;

**int** mid **=** L **+** R **>>** 1;

best[mid] **=** from;

**for** (**int** i **=** from **+** 1; i **<=** to; **++**i)

**if** (eval(mid **+** 1, best[mid]) **>** eval(mid **+** 1, i))

best[mid] **=** i;

solve(L, mid **-** 1, from, best[mid]);

solve(mid **+** 1, R, best[mid], to);

}

Độ phức tạp thuật toán:

* Bộ nhớ: O(n)
* Thời gian: O(n) ⬄ tầng solve, mỗi tầng xét n phần tử

Một số bài tập thực hành:

[[IOI] Guardians of the Lunatics](https://www.hackerrank.com/contests/ioi-2014-practice-contest-2/challenges/guardians-lunatics-ioi14)

[F - ACM ICPC Vietnam Regional 2017](https://open.kattis.com/problems/famouspagoda)

[Hackerrank - Mining](https://www.hackerrank.com/contests/world-codesprint-5/challenges/mining)

**Một số kĩ thuật khác:**

[1.Bao lồi đường thẳng(Convex Hull Trick)](https://vnoi.info/wiki/translate/wcipeg/Convex-Hull-Trick.md)

[2.Tối ưu bằng stack](https://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong.md#4-t%E1%BB%91i-%C6%B0u-b%E1%BA%B1ng-stack)

[3.Quản lí đồ thị hàm quy hoạch động (Slope trick)](https://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong.md#5-qu%E1%BA%A3n-l%C3%AD-%C4%91%E1%BB%93-th%E1%BB%8B-h%C3%A0m-quy-ho%E1%BA%A1ch-%C4%91%E1%BB%99ng-slope-trick)

[4.Lợi dụng tính thưa của bảng QHĐ](https://www.giaithuatlaptrinh.com/?p=152)