# Môn: Tin học

**Chủ đề: QUY HOẠCH ĐỘNG VÀ CÁC BÀI TẬP ỨNG DỤNG**

**MỤC LỤC**

[QUY HOẠCH ĐỘNG VÀ CÁC BÀI TẬP ỨNG DỤNG](#_bookmark0) [4](#_bookmark0)

1. [Mở đầu](#_bookmark1) [4](#_bookmark1)
2. [Một số khái niệm, kiến thức cơ bản](#_bookmark2) [5](#_bookmark2)
   1. [Bài toán tối ưu](#_bookmark3) [5](#_bookmark3)
   2. [Phương pháp chia để trị](#_bookmark4) [5](#_bookmark4)
   3. [Phương pháp Quy hoạch động](#_bookmark5) [7](#_bookmark5)
3. [Một số bài tập áp dụng](#_bookmark6) [14](#_bookmark6)
   1. [Bài tập 1. VM 08 Bài 01 - Bậc thang (Mức độ cơ bản)](#_bookmark7) [14](#_bookmark7)
      1. [Đề bài](#_bookmark8) [14](#_bookmark8)
      2. [Phân tích bài toán](#_bookmark9) [14](#_bookmark9)
      3. [Chương trình minh họa](#_bookmark10) [15](#_bookmark10)
      4. [Test trực tiếp trên link https://oj.vnoi.info/problem/vsteps](#_bookmark11) [15](#_bookmark11)
   2. [Bài tập 2. Dãy con tăng dài nhất (bản dễ) (Mức độ cơ bản)](#_bookmark12) [16](#_bookmark12)
      1. [Đề bài](#_bookmark13) [16](#_bookmark13)
      2. [Phân tích bài toán](#_bookmark14) [16](#_bookmark14)
      3. [Chương trình minh họa](#_bookmark15) [16](#_bookmark15)
      4. [Test trực tiếp trên link https://oj.vnoi.info/problem/liq](#_bookmark16) [17](#_bookmark16)
   3. [Bài tập 3. Atcoder Educational DP Contest A - Frog 1 (Mức độ cơ bản)](#_bookmark17) [17](#_bookmark17)
      1. [Đề bài](#_bookmark18) [17](#_bookmark18)
      2. [Phân tích bài toán](#_bookmark19) [18](#_bookmark19)
      3. [Chương trình minh họa](#_bookmark20) [18](#_bookmark20)
      4. [Test trực tiếp trên link https://oj.vnoi.info/problem/atcoder\_dp\_a](#_bookmark21) [19](#_bookmark21)
   4. [Bài tập 4. Atcoder Educational DP Contest B - Frog 2 (Mức độ cơ bản)](#_bookmark22) [19](#_bookmark22)
      1. [Đề bài](#_bookmark23) [19](#_bookmark23)
      2. [Phân tích bài toán](#_bookmark24) [20](#_bookmark24)
      3. [Chương trình minh họa](#_bookmark25) [20](#_bookmark25)
      4. [Test trực tiếp trên link https://oj.vnoi.info/problem/atcoder\_dp\_b](#_bookmark26) [20](#_bookmark26)
   5. [Bài tập 5. Chọn đồ (Mức độ cơ bản)](#_bookmark27) [21](#_bookmark27)
      1. [Đề bài](#_bookmark28) [21](#_bookmark28)
      2. [Phân tích bài toán](#_bookmark29) [22](#_bookmark29)
      3. [Chương trình minh họa](#_bookmark30) [22](#_bookmark30)
   6. [Bài tập 6. Atcoder Educational DP Contest I - Coins (Mức độ khá)](#_bookmark31) [23](#_bookmark31)
      1. [Đề bài](#_bookmark32) [23](#_bookmark32)
      2. [Phân tích bài toán](#_bookmark33) [24](#_bookmark33)
      3. [Chương trình minh họa](#_bookmark34) [25](#_bookmark34)
      4. [Test trực tiếp trên link https://oj.vnoi.info/problem/atcoder\_dp\_i](#_bookmark35) [25](#_bookmark35)
   7. [Bài tập 7. Atcoder Educational DP Contest Q - Flowers (Mức độ khá)](#_bookmark36) [25](#_bookmark36)
      1. [Đề bài](#_bookmark37) [25](#_bookmark37)
      2. [Phân tích bài toán](#_bookmark38) [26](#_bookmark38)
      3. [Chương trình minh họa](#_bookmark39) [27](#_bookmark39)
   8. [Bài tập 8. Dãy chữ số (Mức độ khá)](#_bookmark40) [28](#_bookmark40)
      1. [Đề bài](#_bookmark41) [28](#_bookmark41)
      2. [Phân tích bài toán](#_bookmark42) [28](#_bookmark42)
      3. [Chương trình minh họa](#_bookmark43) [29](#_bookmark43)
   9. [Bài tập 8. MUA NHÀ (Mức độ khá)](#_bookmark44) [31](#_bookmark44)
      1. [Đề bài](#_bookmark45) [31](#_bookmark45)
      2. [Phân tích bài toán](#_bookmark46) [32](#_bookmark46)
      3. [Chương trình minh họa](#_bookmark47) [32](#_bookmark47)
   10. [Bài tập 9. KINH DOANH (Mức độ khá)](#_bookmark48) [34](#_bookmark48)
       1. [Đề bài](#_bookmark49) [34](#_bookmark49)
       2. [Phân tích bài toán](#_bookmark50) [35](#_bookmark50)
       3. [Chương trình minh họa](#_bookmark51) [35](#_bookmark51)
   11. [Bài tập 10. VẼ BIỂU ĐỒ (Mức độ khó)](#_bookmark52) [37](#_bookmark52)
       1. [Đề bài](#_bookmark53) [37](#_bookmark53)
       2. [Phân tích bài toán](#_bookmark54) [39](#_bookmark54)
       3. [Chương trình minh họa](#_bookmark55) [40](#_bookmark55)
   12. [Bài tập 11. Dãy con tăng dài nhất (Mức độ khó)](#_bookmark56) [44](#_bookmark56)
       1. [Đề bài](#_bookmark57) [44](#_bookmark57)
       2. [Phân tích bài toán](#_bookmark58) [45](#_bookmark58)
       3. [Chương trình minh họa](#_bookmark59) [46](#_bookmark59)
       4. [Test trực tiếp trên link https://oj.vnoi.info/problem/fcc2023\_lis](#_bookmark60) [50](#_bookmark60)
   13. [Bài tập 11. Thi tốt nghiệp (Mức độ khó)](#_bookmark61) [50](#_bookmark61)
       1. [Đề bài](#_bookmark62) [50](#_bookmark62)
       2. [Phân tích bài toán](#_bookmark63) [52](#_bookmark63)
       3. [Chương trình minh họa](#_bookmark64) [53](#_bookmark64)
   14. [Bài tập 13. PHÂN SỐ TĂNG (Mức độ khó)](#_bookmark65) [56](#_bookmark65)
       1. [Đề bài](#_bookmark66) [56](#_bookmark66)
       2. [Phân tích bài toán](#_bookmark67) [57](#_bookmark67)
       3. [Chương trình minh họa](#_bookmark68) [58](#_bookmark68)
4. [Cải tiến bài toán quy hoạch động bằng kỹ thuật chia để trị](#_bookmark69) [60](#_bookmark69)
   1. [Bài tập 14. Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) (Mức độ](#_bookmark70) [khó)](#_bookmark70) [61](#_bookmark70)
      1. [Đề bài](#_bookmark71) [61](#_bookmark71)
      2. [Phân tích bài toán](#_bookmark72) [61](#_bookmark72)
      3. [Chương trình minh họa](#_bookmark73) [64](#_bookmark73)
   2. [Bài tập 15. SEQPART (Mức độ khó)](#_bookmark74) [66](#_bookmark74)
      1. [Đề bài](#_bookmark75) [66](#_bookmark75)
      2. [Phân tích bài toán](#_bookmark76) [67](#_bookmark76)
      3. [Chương trình minh họa](#_bookmark77) [67](#_bookmark77)
5. [Một số bài khác](#_bookmark78) [69](#_bookmark78)
6. [Kết luận](#_bookmark79) [69](#_bookmark79)
7. [Nguồn tài liệu tham khảo](#_bookmark80) [69](#_bookmark80)

# QUY HOẠCH ĐỘNG VÀ CÁC BÀI TẬP ỨNG DỤNG

# Mở đầu

Trong các kỳ thi học sinh giỏi tin học như: Học sinh giỏi khu vực duyên hải; Học sinh giỏi quốc gia; Olympic tin học quốc tế; … các lớp bài toán về tối ưu hóa luôn được ưu tiên lựa chọn trong các đề thi, vì tính ứng dụng vào thực tiễn cao.

Thuật toán Quy hoạch động (Dynamic Programming) là một công cụ hữu ích trong lập trình thi đấu, đặc biệt là trong các bài toán tối ưu hóa. Điểm mạnh của thuật toán này là nó có thể giải quyết các bài toán phức tạp một cách hiệu quả và chính xác. Số lượng bài toán tin học được giải bằng phương pháp quy hoạch động rất lớn, số lượng các bài thi có thể áp dụng phương pháp quy hoạch động để giải trong đề thi học sinh giỏi môn Tin học thường có điểm rất cao. Vậy *“Có phải tất cả các bài toán tối ưu đều có thể áp dụng phương pháp quy hoạch động để giải?; “Làm thế nào để nhận dạng được bài toán đó có thể áp dụng phương pháp quy hoạch động để giải?; “Làm thế nào có thể giải một bài toán bằng phương pháp quy hoạch động?”*;

Có nhiều chiến lược để thiết kế giải thuật: Chia để trị (divide and conquer); Đệ quy quay lui (backtracking); Quy hoạch động (dynamic programming); Tham lam (greedy strategy), … Mỗi chiến lược phù hợp cho một số dạng bài toán, mỗi chiến lược có ưu và nhược điểm riêng; Nhiệm vụ: cần quyết định chọn chiến lược phù hợp để giải quyết bài toán đó một cách hợp lí.

Chuyên đề này, tôi xin phép chia sẻ một số bài tập có thể áp dụng phương pháp quy hoạch động trong quá trình dạy đội tuyển HSG môn Tin học lớp 10 trong hè chuẩn bị lên lớp 11.

Nhân đây, tôi cũng xin gửi lời cảm ơn tới các thầy cô, các bạn đồng nghiệp đã giúp đỡ, cung cấp tài liệu và góp ý để tôi bổ sung, hoàn thiện chuyên đề. Hi vọng rằng chuyên đề sẽ cung cấp cho các bạn đồng nghiệp và các em học sinh một phần kiến thức bổ ích.

# Link tải bộ TEST của các bài tập:

*[https://drive.google.com/file/d/1MHmaCl1bjq0trtxDZT8o8Jzeh-](https://drive.google.com/file/d/1MHmaCl1bjq0trtxDZT8o8Jzeh-xqPPaT/view?usp=sharing) [xqPPaT/view?usp=sharing](https://drive.google.com/file/d/1MHmaCl1bjq0trtxDZT8o8Jzeh-xqPPaT/view?usp=sharing)*

# Một số khái niệm, kiến thức cơ bản

# Bài toán tối ưu

## \* Khái niệm

Bài toán tối ưu gồm có 1 hàm f gọi là hàm mục tiêu hay hàm đánh giá; các hàm g1, g2, …, gn cho giá trị logic gọi là hàm ràng buộc. Yêu cầu của bài toán là tìm một cấu hình x thoả mãn tất cả các ràng buộc g1, g2, …, gn: gi(x) = TRUE (*i* :1  *i*  *n*)

và x là tốt nhất, theo nghĩa không tồn tại một cấu hình y nào khác thoả mãn các hàm ràng buộc mà f(y) tốt hơn f(x).

*Ví dụ 1:* Trong mặt phẳng tọa độ Oxy tìm tọa độ (x,y) để tổng x + y đạt giá trị lớn nhất mà x2 + y2 ≤ 1.

Ở bài toán trên ta thấy:

Hàm mục tiêu: x + y  max Hàm ràng buộc: x2 +y2 ≤ 1.

*Ví dụ 2:* Bài toán xếp Ba lô Có một ba lô có thể chứa tối đa trọng lượng M và có n đồ vật (n ≤ 100), mỗi đồ vật có trọng lượng wi và giá trị bi; M, wi, bi là các số nguyên. Hãy chọn và xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất.

Với bài toán trên ta thấy:

Hàm mục tiêu: Hàm ràng buộc:

*bi*  max, i=1,2, ...,n

*wi*  *M* , i=1,2, ..., n

Tóm lại, bài toán tối ưu rất phong phú, đa dạng, được ứng dụng nhiều trong thực tế nhưng chúng ta cũng cần biết rằng đa số các bài toán tối ưu là không giải được hoặc chưa giải được.

# Phương pháp chia để trị

*Ý tưởng :*

+ Chia vấn đề cần giải thành một số vấn đề con cùng dạng với vấn đề đã cho, chỉ khác là cỡ của chúng nhỏ hơn.

+ Mỗi vấn đề con được giải quyết độc lập, nếu vấn đề con chưa tìm được nghiệm ngay thì lại tiếp tục chia nhỏ theo tư tưởng như trên.

+ Sau đó, ta kết hợp nghiệm của các vấn đề con để nhận được nghiệm của vấn đề đã cho.

Chia để trị là một trong những phương pháp hiệu để thiết kế thuật toán. Nguyên lý thực hiện của thuật toán chia để trị thực hiện qua hai bước sau:

* Bước 1 (chia): Chia bài toán lớn thành các bài toán con nhỏ hơn
* Bước 2 (trị): Gọi đệ quy giải các bài toán con, sau đó gộp lời giải của bài toán con thành lời giải bài toán lớn.

Đối với nhiều thuật toán đệ quy chúng ta đã tìm hiểu, nguyên lý chia để trị (devide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán.

*Ví dụ:* Bài toán Tính dãy Fibonacci

Ta có định nghĩa như sau:

*F* *n*

 1 (*n*  0 **or** *n*  1)



*F* (*n* 1)  *F* (*n*  2) (*n*  2)

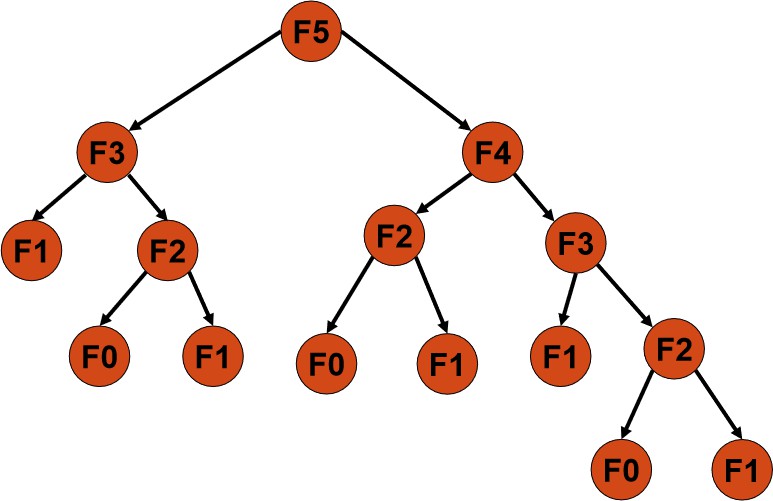
Một số phần tử đầu tiên của dãy Fibonacci: 0, 1, 1, 2, 3, 5, 8, … Tìm số Fibonacci thứ n.

**Dữ liệu vào:** Một số nguyên dương duy nhất n (n ≤ 1000000).

**Kết quả ra:** Một số nguyên duy nhất là số Fibonacci thứ n (kết quả lấy phần dư cho 1000000007).

*Ví dụ:*

|  |  |
| --- | --- |
| FIBO.INP | FIBO.OUT |
| 5 | 8 |
| 20 | 6765 |

**

# Phương pháp Quy hoạch động

* + 1. **Bài toán quy hoạch động**

Như chúng ta đã biết, một bài toán có thể giải được bằng phương pháp quy hoạch động cần đảm bảo 2 đặc điểm nổi bật sau:

* + - * Có tính chất của các bài toán con gối nhau (overlapping subproblem): Có thể chia nhỏ một bài toán lớn thành các bài toán con.
      * Có cấu trúc con tối ưu (optimal substructure): Kết hợp lời giải của các bài toán con ta được lời giải của bài toán lớn hơn.

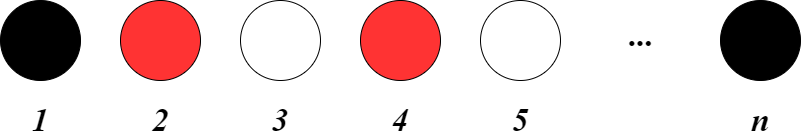
Để mở đầu ta xét ví dụ sau:

***Ví dụ:*** Bạn An có n chiếc ghế màu trắng, n chiếc ghế màu đen và n chiếc ghế màu đỏ. An muốn chọn ra n chiếc ghế để xếp thành một hàng ngang. Do An không thích màu đỏ nên An không muốn xếp hai chiếc ghế đỏ cạnh nhau. Tính số cách xếp ghế thỏa mãn điều kiện đó.

Điều kiện: 1  *n*  105.

***Lưu ý****: hai cách xếp được xem là khác nhau khi tồn tại một vị trí mà hai cách có hai loại ghế khác nhau.*

*Ví dụ:*



Đỏ

Trắng

Đỏ

Trắng

Đen

Đen

Ta xây dựng thuật giải như sau:

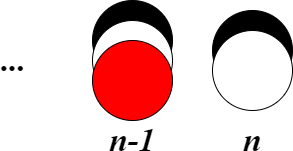
# Thuật toán đệ quy

Gọi số cách xếp i cái ghế là *f* [*i*] . Ta xét chiếc ghế thứ n.

- Nếu nó có màu đen hoặc trắng thì chiếc ghế cạnh nó có thể có một trong ba màu. Do đó ta chỉ cần bố trí n-1 chiếc ghế còn lại thỏa mãn yêu cầu. Do có 2 cách

chọn màu cho ghế thứ n và *f* [*n* 1] cách chọn màu cho các ghế còn lại nên số cách

xếp trong trường hợp này là 2 \* *f* [*n* 1] .

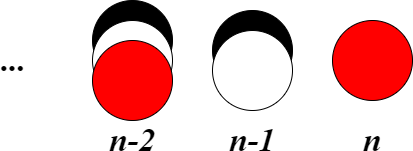


Nếu nó có màu đỏ thì chiếc ghế cạnh nó chỉ có thể có màu trắng hoặc đen. Do vậy

nên chiếc ghế thứ *n*  2 có thể có một trong ba màu. Khi đó ta cũng chỉ cần bố trí

*n*  2 chiếc ghế còn lại thỏa mãn yêu cầu. Số cách xếp trong trường hợp này là

1\* 2 \* *f* [*n*  2].



Với ý tưởng trên, ta có thể giải bài toán này như các bài toán đệ quy đơn giản. Cài đặt như sau:

// Tính số cách sắp xếp n cái ghế int solve(int n)

{

// Trường hợp cơ bản if (n == 1)

return 3; if (n == 2) return 8;

// Bước đệ quy

return 2 \* solve(n - 1) + 2 \* solve(n - 2);

}

Thuật toán trên có độ phức tạp lũy thừa nên chỉ áp dụng được với n nhỏ(*n*  45)

, không đủ nhanh so với yêu cầu bài toán.

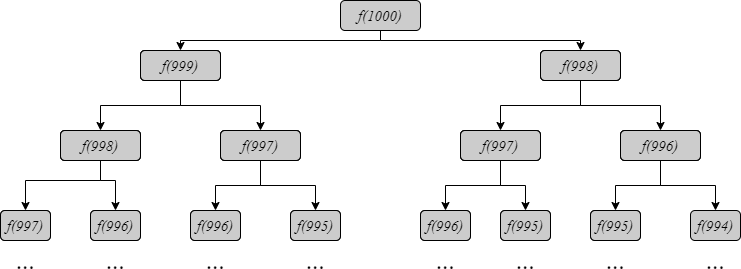
# Tối ưu thuật toán đệ quy

Thuật toán trên chạy chậm vì một số hàm solve(i) được gọi rất nhiều lần. Ta lấy ví dụ sau:

Giả sử cần tính solve(1000). Khi đó cần tính solve(999) và solve(998).

Để tính solve(999) cần gọi hàm solve(998) và solve(997). Để tính solve(998) cần gọi hàm solve(997) và solve(996). Để tính solve(997) cần gọi hàm solve(996) và solve(995).

Ta có thể biểu diễn các hàm được gọi bằng một sơ đồ như sau:



Từ sơ đồ trên ta thấy có nhiều hàm bị gọi rất nhiều lần một cách không cần thiết: solve(998) được gọi 2 lần

solve(997) được gọi 3 lần solve(996) được gọi 5 lần

Để khắc phục điều này ta có thể sử dụng một mảng nhớ d sao cho solve(i):

Cài đặt như sau:

*d*[*i*] là giá trị của

int d[100010]; int solve(int n)

{

if (n == 1) return 3; else if (n == 2) return 8;

else if (d[n] != 0) return d[n]; else

{ d[n] = 2 \* f(n - 1) + 2 \* f(n - 2); return d[n]; }

}

Thuật toán trên có độ phức tạp *O*(*n*) .

Với cách tiếp cận trên, ta quan tâm đến giá trị cuối cùng xét những giá trị bé hơn cần thiết cho tính toán.

*f* [*n*], sau đó mới xem

Nhưng với phương pháp quy hoạch động, ta sẽ quan tâm đến các bài toán với tham số nhỏ hơn trước tiên.

# Các khái niệm

* + - * Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động.
      * Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động.
      * Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động.
      * Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Phương pháp quy hoạch động (Dynamic Programming) là một kỹ thuật nhằm đơn giản hóa việc tính toán các công thức truy hồi bằng cách lưu toàn bộ hay một phần kết quả tính toán tại mỗi bước trước đó với mục đích sử dụng lại.

Như vậy, Quy hoạch động = Chia để trị + Mảng (lưu lại kết quả).

Phương pháp quy hoạch động do nhà toán học người Mỹ Richard Bellman (1920-1984) phát minh năm 1953. Phương pháp này dùng để giải các bài toán tối ưu có bản chất đệ qui, tức là tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

## Điểm khác nhau cơ bản giữa quy hoạch động và phương pháp đệ quy là :

+ Phương pháp đệ quy giải quyết bài toán theo hướng topdown, nghĩa là để giải bài toán ban đầu, ta phải đi giải tất cả các bài toán con của nó. Đây là một phương pháp hay, tuy nhiên phương pháp này sẽ gặp hạn chế về mặt thời gian, tốc độ do phải tính đi tính lại nhiều lần một số bài toán con giống nhau nào đó.

+ Phương pháp quy hoạch động sử dụng nguyên lý bottom-up, nghĩa là "đi từ dưới lên". Đầu tiên, ta sẽ phải 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 và 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 được tốc độ.

# Phương pháp tiếp cận Khi nào có thể áp dụng QHĐ

Quy hoạch động được sử dụng khi ta tìm được công thức liên hệ giữa kết quả bài toán có đầu vào cho trước với một (hoặc một số) bài toán con tương tự nhưng có đầu vào nhỏ hơn. Khi ta biết được một số trạng thái bắt đầu của bài toán, nói cách khác - bài toán con với những đầu vào rất nhỏ, ta có thể sử dụng QHĐ để tính ra kết quả cuối cùng.

# Trạng thái của bài toán là gì?

Trạng thái là một trường hợp, một bài toán con của bài toán lớn với tham số cho trước.

Ví dụ, trạng thái trong ví dụ là số cách sắp xếp n chiếc ghế thỏa mãn không có hai ghế đỏ cạnh nhau.

# Liên hệ giữa các trạng thái

Để giải bài toán quy hoạch động, điều quan trọng nhất là tìm ra mối liên hệ giữa một trạng thái và các trạng thái có tham số nhỏ hơn.

Gọi *f* [*i*] là cách sắp xếp i chiếc ghế thành một hàng dọc. Khi đó ta có:

 *f* [1]  3; *f* [2]  8

 *f* [*i*]  2 *f* [*i* 1]  2 *f* [*i*  2],*i*  3;4;...;*n*(\*)



Công thức (\*) được gọi là ***công thức truy hồi***.

# Tìm kết quả cuối cùng

Sau khi đã biết công thức truy hồi và tính được

# Chương trình tham khảo

*f* [1],

*f* [2], ta có thể tìm

*f* [*n*].

#include <bits/stdc++.h> using namespace std; long long n, f[100010]; int main()

{

cin >> n;

f[1] = 3; f[2] = 8;

for (int i = 3; i <= n; i++)

f[i] = 2 \* f[i - 1] + 2 \* f[i - 2]; cout << f[n];

return 0;

}

Độ phức tạp của thuật toán trên là *O*(*n*) , nhưng cách thực hiện đơn giản hơn đệ quy có nhớ.

# Qui trình giải toán bằng Quy hoạch động

* **Các bước thực hiện thuật toán quy hoạch động như sau:**
  1. Xác định và phân tích bài toán: Xác định bài toán cần giải quyết, cấu trúc của bài toán, đặc điểm và các điều kiện của bài toán.
  2. Xác định phương pháp tối ưu: Xác định cách tối ưu hóa và tìm kiếm phương án tối ưu cho bài toán.
  3. Xác định bài toán con (sub-problems): Chia bài toán ban đầu thành các bài toán con nhỏ hơn, có cùng cấu trúc và có thể giải quyết bằng các phương pháp tối ưu hóa.
  4. Xây dựng bảng quy hoạch động: Tạo ra bảng quy hoạch động, một ma trận lưu trữ kết quả của các bài toán con và kết quả tối ưu của bài toán ban đầu.
  5. Thiết lập công thức đệ quy: Thiết lập công thức đệ quy để tính toán kết quả tối ưu của các bài toán con, dựa trên kết quả đã tính toán trước đó.
  6. Tính toán kết quả tối ưu của bài toán ban đầu: Tính toán kết quả tối ưu của bài toán ban đầu dựa trên bảng quy hoạch động và công thức đệ quy đã thiết lập ở bước trước.
  7. Truy vết kết quả: Nếu cần, thực hiện truy vết để tìm ra phương án tối ưu của bài toán ban đầu từ bảng quy hoạch động.
  8. Kiểm tra và tối ưu hóa: Kiểm tra kết quả và tối ưu hóa thuật toán nếu cần thiết. Các bước trên có thể thực hiện theo thứ tự khác nhau tùy vào bài toán cụ thể,

tuy nhiên cần đảm bảo bước đệ quy được thiết lập đúng để đảm bảo tính chính xác

và hiệu quả của thuật toán quy hoạch động.

# Cấu trúc của thuật toán quy hoạch động bao gồm các phần sau:

* 1. Bảng quy hoạch động: Đây là một ma trận lưu trữ kết quả của các bài toán con và kết quả tối ưu của bài toán ban đầu. Bảng quy hoạch động được xây dựng bằng cách khởi tạo một ma trận có kích thước phù hợp với bài toán và tính toán kết quả tối ưu cho các bài toán con.
  2. Hàm truy hồi (recurrence function): Đây là hàm đệ quy tính toán kết quả tối ưu của các bài toán con dựa trên kết quả đã tính toán trước đó. Hàm truy hồi được thiết lập sao cho nó tính toán kết quả tối ưu của bài toán con dựa trên các kết quả tối ưu của các bài toán con nhỏ hơn.
  3. Các bước cập nhật bảng quy hoạch động: Để tính toán kết quả tối ưu của bài toán ban đầu, thuật toán quy hoạch động sẽ thực hiện các bước cập nhật bảng quy

hoạch động. Các bước này bao gồm tính toán các kết quả tối ưu của các bài toán con, và cập nhật kết quả tối ưu của bài toán ban đầu dựa trên các kết quả tối ưu của các bài toán con nhỏ hơn.

* 1. Truy vết kết quả: Nếu cần, thuật toán quy hoạch động sẽ thực hiện truy vết để tìm ra phương án tối ưu của bài toán ban đầu từ bảng quy hoạch động.

Các phần này là các thành phần cơ bản của thuật toán quy hoạch động và có thể được sử dụng để giải quyết nhiều bài toán khác nhau. Tuy nhiên, cấu trúc chi tiết của thuật toán quy hoạch động có thể khác nhau tùy thuộc vào bài toán cụ thể.

# Các phương pháp tối ưu hóa trong thuật toán quy hoạch động

1. Trong thuật toán quy hoạch động, có nhiều phương pháp tối ưu hóa khác nhau để tính toán kết quả tối ưu của bài toán. Dưới đây là một số phương pháp tối ưu hóa thường được sử dụng trong thuật toán quy hoạch động:
2. Cắt ngắn chiều dài của bảng quy hoạch động: Phương pháp này giúp giảm bớt thời gian và bộ nhớ cần thiết để tính toán kết quả tối ưu bằng cách loại bỏ các giá trị không cần thiết trong bảng quy hoạch động. Điều này có thể được thực hiện bằng cách giới hạn kích thước của bảng quy hoạch động hoặc chỉ tính toán một phần của bảng quy hoạch động.
3. Tối ưu hoá thứ tự tính toán: Phương pháp này giúp tối ưu hoá thứ tự tính toán các bài toán con để giảm bớt số lần tính toán trùng lặp và tăng tốc độ tính toán. Thông thường, các bài toán con nên được tính toán theo thứ tự từ nhỏ đến lớn để tối ưu hoá thời gian tính toán.
4. Sử dụng kỹ thuật tạm thời (Memoization): Kỹ thuật tạm thời (Memoization) là một phương pháp lưu trữ kết quả của các bài toán con đã tính toán để tránh tính toán lại trong quá trình tính toán kết quả tối ưu của bài toán lớn hơn. Kỹ thuật này giúp giảm thiểu số lần tính toán trùng lặp và tăng tốc độ tính toán.
5. Sử dụng kỹ thuật tối ưu hóa không gian đệ quy: Kỹ thuật tối ưu hóa không gian đệ quy giúp giảm bớt bộ nhớ cần thiết để tính toán kết quả tối ưu bằng cách sử dụng một ngăn xếp (stack) thay vì đệ quy để lưu trữ các kết quả của các bài toán con.
6. Những phương pháp tối ưu hóa này giúp cải thiện hiệu suất tính toán của thuật toán quy hoạch động và giúp tìm ra kết quả tối ưu của bài toán nhanh hơn và chính xác hơn.

# Một số bài tập áp dụng

# Bài tập 1. VM 08 Bài 01 - Bậc thang (Mức độ cơ bản)

*<https://oj.vnoi.info/problem/vsteps>*

* + 1. Đề bài

Bờm chơi trò chơi điện tử Lucky Luke đến màn phải điều khiển Lucky leo lên một cầu thang gồm 𝑁 bậc.

Các bậc thang được đánh số từ 1 đến 𝑁 từ dưới lên trên. Lucky có thể đi lên một bậc thang, hoặc nhảy một bước lên hai bậc thang. Tuy nhiên một số bậc thang đã bị thủng do cũ kỹ và Lucky không thể bước chân lên được. Biết ban đầu, Lucky đứng ở bậc thang số 1 (bậc thang số 1 không bao giờ bị thủng).

Chơi đến đây, Bờm chợt nảy ra câu hỏi: có bao nhiêu cách để Lucky leo hết được cầu thang? (Nghĩa là leo đến bậc thang thứ 𝑁). Bờm muốn nhờ bạn trả lời câu hỏi này.

# Dữ liệu vào:

Dòng đầu tiên: gồm 2 số nguyên 𝑁 và 𝐾, là số bậc của cầu thang và số bậc thang bị hỏng (0≤𝐾<𝑁≤105).

Dòng thứ hai: gồm 𝐾 số nguyên cho biết chỉ số của các bậc thang bị hỏng theo thứ tự tăng dần.

**Kết quả ra:** In ra phần dư của số cách Lucky leo hết cầu thang khi chia cho 14062008.

# Ví dụ:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 4 2  2 3 | 0 |
| 90000 1  49000 | 4108266 |

* + 1. Phân tích bài toán

Giải thuật [Quy hoạch động](https://vnspoj.github.io/category/dp) đơn giản như sau:

Trước tiên tạo mạng lưu trạng thái của một bậc thang xem nó sống hay chết. Ta dùng mảng *yes[i]*. Nếu *yes[i] = true* thì không hỏng và ngược lại. Thứ hai, ta gọi *F[i]* là số cách leo lên bậc thang thứ *i.* Đương nhiên nếu nó hỏng thì chẳng có cách nào để leo lên và *f[i] = 0.*

Ban đầu *f[1] = 1* vì nó đang đứng ở đó*. F[2]* tùy thuộc nó chết hay sống mà *f[2] 0 or 1*. Xét tại một bước thứ *i*. Nếu như không có đường đến nó thì *f[i] = 0.* Ngược

lại nếu trước đó bậc *i-1* chết thì bậc *i* không thể lên từ *i-1,* bậc *i-2* sống thì ta có một con đường duy nhất lên *i* là từ *i-2* và *f[i] = f[i-2]* else nếu *i-1* sống và *i-2* chết thì cũng vậy *f[i] = f[i-1]* else nữa cả hai cùng sống thì *f[i] = f[i-1] + f[i-2]*. Nếu cả hai cùng chết thì đương nhiên chẳng có đường nào lên nổi *n.*

Độ phức tạp: O(N).

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std;

const long base = 14062008; long n,k,t,f[150000];

bool yes[150000] = {true}; int main()

{

scanf("%li %li",&n,&k);

for (long i=0;i<=n;i++) yes[i] = true; for (long i=1;i<=k;i++)

{

scanf("%li",&t); yes[t] = false;

}

f[0] = 0;

f[1] = 1;

if (yes[2]) f[2] = 1; else f[2] = 0; for (long i=3;i<=n;i++)

if (not(yes[i])) f[i] = 0; else

if ((not(yes[i-1])) and (yes[i-2])) f[i] = f[i-2] % base; else

if ((not(yes[i-2])) and (yes[i-1])) f[i] = f[i-1] % base; else f[i] = (f[i-1] % base + f[i-2] % base) % base;

printf("%li",f[n]); return 0;

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/vsteps>*

# Bài tập 2. Dãy con tăng dài nhất (bản dễ) (Mức độ cơ bản)

*<https://oj.vnoi.info/problem/liq>*

* + 1. Đề bài

Cho một dãy số nguyên gồm 𝑁 phần tử Biết rằng dãy con tăng đơn điệu là 1 dãy

*A*1, *A*2 ,..., *AN* .

*Ai*1, *Ai* 2 ,..., *Aik*

thỏa

mãn 𝑖1<𝑖2<⋯<𝑖𝑘 và *Ai*1  *Ai*2  ...  *Aik* . Hãy cho biết dãy con tăng đơn điệu dài

nhất của dãy này có bao nhiêu phần tử?

## Dữ liệu vào:

* Dòng 1 gồm 1 số nguyên là số 𝑁(1≤𝑁≤1000).
* Dòng thứ 2 ghi 𝑁 số nguyên *A* , *A* ,..., *A* (1 *A* 104) .

1 2 *N i*

***Kết quả ra:*** Ghi ra độ dài của dãy con tăng đơn điệu dài nhất.

# Ví dụ:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 6  1 2 5 4 6 2 | 4 |

***Giải thích:*** Dãy con dài nhất là dãy là 4.

* + 1. Phân tích bài toán

*A*1 1 *A*2  2  *A*4  4  *A*5  6, độ dài dãy này

Gọi 𝐹(𝑖) là dãy con tăng dài nhất kết thúc ở 𝐴(𝑖), ta có công thức tính: F(1)=1

F(i)=max{F(j)+1}

Với j thỏa 1≤j<𝑖 và A(j)<A(i). Kết quả bài toán là max{F}.

Bài này N có giới hạn nhỏ nên có thể dùng 2 vòng for để tính, có thể cài đặt công thức trên với độ phức tạp O(N2).

* + 1. Chương trình minh họa

#include <iostream> #include <vector> using namespace std; int main() {

ios::sync\_with\_stdio(false); cin.tie(0); int n; cin >> n;

vector<int> a(n);

for (int &x: a) cin >> x; vector<int> f(n);

int result = 1;

for (int i=0; i<n; i++) { f[i] = 0;

for (int j=i-1; j>=0; j--) if (a[i] > a[j]) {

f[i] = max(f[i], f[j]);

}

f[i] += 1;

result = max(result, f[i]);

}

cout << result; return 0;

}

* + 1. Test trực tiếp trên link <https://oj.vnoi.info/problem/liq>

# Bài tập 3. Atcoder Educational DP Contest A - Frog 1 (Mức độ cơ bản)

*<https://oj.vnoi.info/problem/atcoder_dp_a>*

* + 1. Đề bài

Có 𝑁 hòn đá, được đánh số từ 1 đến 𝑁. Với mỗi chỉ số 𝑖 (1≤𝑖≤𝑁), độ cao của hòn đá thứ 𝑖 là ℎ𝑖. Ban đầu, có một chú ếch đang ngồi ở hòn đá thứ nhất và chú sẽ thực hiện liên tục một loạt các hành động sau:

Nếu chú đang ngồi ở hòn đá 𝑖 chú có thể nhảy đến hòn đá thứ 𝑖+1 hoặc 𝑖+2.

Chú sẽ mất chi phí khi nhảy là *hi*  *hj* với 𝑗 là hòn đá mà chú ếch nhảy đến.

Bạn hãy giúp chú ếch tìm chi phí tối thiểu để nhảy từ hòn đá thứ nhất đến hòn đá thứ 𝑁 nhé.

## Dữ liệu vào:

* Dòng đầu tiên của dữ liệu vào chứa số nguyên dương 𝑁 (2≤𝑁≤105), là số lượng hòn đá.
* Dòng thứ hai gồm 𝑁 số nguyên ℎ𝑖 (1≤𝑖≤𝑁,1≤ℎ𝑖≤104), là độ cao của hòn đá thứ 𝑖.

***Kết quả ra:*** Gồm một số nguyên, là chi phí ít nhất để nhảy từ hòn đá thứ nhất đến hòn đá thứ 𝑁.

## Ví dụ:

|  |  |  |
| --- | --- | --- |
| **INPUT** | **OUTPUT** | **Giải thích** |
| 4  10 30 40 20 | 30 | Một đường đi tối ưu là: 1→2→4. Chi phí sẽ là: |10−30|+|30−20|=30. |
| 2  10 10 | 0 | Một đường đi tối ưu là: 1→2. Chi phí sẽ là: |10−10|=0. |
| 6  30 10 60 10 60 50 | 40 | Một đường đi tối ưu là: 1→3→5→6. Chi phí sẽ là:  |30−60|+|60−60|+|60−50|=40. |

* + 1. Phân tích bài toán

Để giải quyết bài này ta sẽ dùng quy hoạch động. Ta định nghĩa 𝑑𝑝[𝑖] là chi phí ít nhất để đến được hòn đá thứ 𝑖. Dễ thấy để đến được hòn đá thứ 𝑖, ta chỉ có thể nhảy từ:

* Hòn đá thứ 𝑖−1 với (1<𝑖≤𝑁).
* hòn đá thứ 𝑖−2 với (2<𝑖≤𝑁).

Vì chú ếch bắt đầu từ hòn đá đầu tiên nên: 𝑑𝑝[1]=0. Từ đây, để đến được hòn đá thứ 2 ta có thể nhảy từ hòn đá đầu tiên ⇒𝑑𝑝[2]=𝑑𝑝[1]+|ℎ[2]−ℎ[1]|. Tính trước 𝑑𝑝[1] và 𝑑𝑝[2] ta có:

* ∀𝑖,2<𝑖≤𝑁,𝑑𝑝[𝑖]=𝑚𝑖𝑛(𝑑𝑝[𝑖−1]+|ℎ[𝑖]−ℎ[𝑖−1]|,𝑑𝑝[𝑖−2]+|ℎ[𝑖]−ℎ[𝑖−2]|) Vì kết thúc ở hòn đá thứ 𝑁 nên kết quả bài toán là 𝑑𝑝[𝑁].

Độ phức tạp thời gian: O(N).

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std;

const long long N = 2e5 + 5; int dp[N], h[N], n;

int main(){ cin >> n;

for (int i = 1; i <= n; i ++) cin >> h[i]; dp[1] = 0; dp[2] = abs(h[1] - h[2]); for (int i = 3; i <= n; i ++)

dp[i] = min(dp[i-1] + abs(h[i] - h[i-1]), dp[i-2] + abs(h[i] - h[i-2])); cout << dp[n];

return 0;

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/atcoder_dp_a>*

# Bài tập 4. Atcoder Educational DP Contest B - Frog 2 (Mức độ cơ bản)

*<https://oj.vnoi.info/problem/atcoder_dp_b>*

* + 1. Đề bài

Có 𝑁 hòn đá, được đánh số từ 1 đến 𝑁. Với mỗi chỉ số 𝑖 (1≤𝑖≤𝑁), độ cao của hòn đá thứ 𝑖 là ℎ𝑖.

Ban đầu, có một chú ếch đang ngồi ở hòn đá thứ nhất và chú sẽ thực hiện liên tục một loạt các hành động sau:

Nếu chú đang ngồi ở hòn đá 𝑖 chú có thể nhảy đến các hòn đá

thứ 𝑖+1, 𝑖+2 ... 𝑖+𝐾. Chú sẽ mất phí khi nhảy là *hi*  *hj* với 𝑗 là hòn đá mà chú ếch nhảy đến.

Bạn hãy giúp chú ếch tìm chi phí tối thiểu để nhảy từ hòn đá thứ nhất đến hòn

đá thứ 𝑁 nhé.

# Dữ liệu vào:

* Dòng đầu tiên của dữ liệu vào chứa hai số nguyên dương 𝑁 và 𝐾 (2≤𝑁≤105,1≤𝐾≤100), lần lượt là số lượng hòn đá và giới hạn nhảy của ếch.
* Dòng thứ hai gồm 𝑁 số nguyên ℎ𝑖 (1≤𝑖≤𝑁,1≤ℎ𝑖≤104), là độ cao của hòn đá thứ 𝑖.

**Kết quả ra**: Gồm một số nguyên, là chi phí ít nhất để nhảy từ hòn đá thứ nhất đến hòn đá thứ 𝑁.

# Ví dụ:

|  |  |  |
| --- | --- | --- |
| **INPUT** | **OUTPUT** | **Giải thích** |
| 5 3  10 30 40 50 20 | 30 | Một đường đi tối ưu là: 1→2→5.  Chi phí sẽ là: |10−30|+|30−20|=30. |
| 3 1  10 20 10 | 20 | Một đường đi tối ưu là: 1→2→3.  Chi phí sẽ là: |10−20|+|20−10|=20. |
| 2 100  10 10 | 40 | Một đường đi tối ưu là: 1→2.  Chi phí sẽ là: |10−10|=0. |
| 10 4  40 10 20 70 80 10 20 70 80 60 | 40 | Một đường đi tối ưu là: 1→4→8→10.  Chi phí sẽ là:  |40−70|+|70−70|+|70−60|=40. |

* + 1. Phân tích bài toán

Để giải quyết bài này ta sẽ dùng quy hoạch động. Ta định nghĩa 𝑑𝑝[𝑖] là chi phí ít nhất để đến được hòn đá thứ 𝑖.

* Ban đầu, chú ếch ở hòn đá đầu tiên nên ta có: 𝑑𝑝[1]=0
* Để đến hòn đá 𝑖 ta có thể đến được từ các hòn đá *i*  *j*,*j* 1  *j*  *K*



1  *i*  *j*

* Cuối cùng, ta có 𝑑𝑝[𝑖] được tính theo công thức sau:

*dp*[*i*] 

min

1 *j**K* ,1*i* *j*

{*dp*[*i* 

*j*]  *h*[*i*]  *h*[*i* 

*j*]}

Vì kết thúc ở hòn đá thứ 𝑁 nên kết quả bài toán là 𝑑𝑝[𝑁]. Độ phức tạp thời gian: O(NK).

* + 1. Chương trình minh họa

#include <iostream> using namespace std;

const long long N = 2e5 + 5; int dp[N], h[N], n, k;

int main(){

cin >> n >> k;

for (int i = 1; i <= n; i ++) cin >> h[i];

dp[1] = 0;

for (int i = 2; i <= n; i ++){ int min\_of\_al\_J = 1e9; for (int j = 1; j <= k; j ++)

if (i - j >= 1)

min\_of\_al\_J = min(min\_of\_al\_J, dp[i - j] + abs(h[i] - h[i - j]));

dp[i] = min\_of\_al\_J;

}

cout << dp[n]; return 0;

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/atcoder_dp_b>*
  1. **Bài tập 5. Chọn đồ (Mức độ cơ bản)**
     1. Đề bài

Hùng cùng các bạn đội tuyển Tin học được đi học tập với các giáo sư đầu ngành về Công nghệ thông tin trong 1 tuần tại Đà Nẵng. Hùng phấn khởi chuẩn bị vali để chứa nhiều đồ dùng cá nhân. Hùng có n đồ dùng cần mang đi, đồ thứ i có trọng lượng là Wi và giá trị Vi. Khi làm thủ tục lên máy bay (check in) ở sân bay, nhân viên thông báo hành lý xách tay vượt quá trọng lượng cho phép nên yêu cầu Hùng phải chọn một số đồ dùng thật cần thiết để lại trong vali, số còn lại chuyển sang hành lý dạng ký gửi nhằm đảm bảo trọng lượng hành lý xách tay theo quy định và an toàn cho chuyến bay. Hùng đang loay hoay tìm cách chọn đồ dùng sao cho hợp lí, các bạn lập trình viên hãy giúp Hùng nhé.

**Yêu cầu:** Hãy cho biết Hải cần chọn những đồ dùng cần thiết nào để lại trong vali

**Dữ liệu vào**: Đọc từ file CHONDO.INP gồm:

* Dòng 1: Chứa hai số nguyên n và m lần lượt là số đồ dùng và trọng lượng

hành lý xách tay cho phép mang lên máy bay *n*  1000, *m*  10000.

* n dòng tiếp theo, mỗi dòng chứa hai số nguyên dương *Wi* ,*Vi* *Wi* , *Vi* 10000

lần lượt là trọng lượng và giá trị sử dụng của đồ vật thứ i.

**Dữ liệu ra**: Ghi vào file CHONDO.OUT gồm:

* Dòng 1: Tổng giá trị sử dụng lớn nhất của các đồ vật được chọn để lại trong vali.
* Dòng 2: Ghi tổng trọng lượng các đồ vật được chọn để lại trong vali.

***Ví dụ:***

|  |  |
| --- | --- |
| **CHONDO.INP** | **CHONDO.OUT** |
| 3 10  3 2  2 6  7 4 | 10  9 |

# Giới hạn:

* Subtask 1: 50% số test ứng với 50% số điểm có *n* 100; *m* 100;*wi* , *vi*  100.
* Subtask 2: 50% số test ứng với 50% số điểm không có ràng buộc bổ sung.
  + 1. Phân tích bài toán

# Tham số thể hiện kích thước bài toán

* + - * Kết quả bài toán là tổng giá trị lớn nhất của các món hàng được chọn trong n món sao cho tổng khối lượng không lớn hơn M cho trước, ký hiệu làF(n)
      * Tham số thể hiện kích thước bài toán làsố món hàng n;
      * Giá trị của F(n) có thể được tính từ giá trị của F(n-1) cộng thêm hoặc không cộng thêm giá trị của món hàng thứ n nhưng tổng khối lượng không lớn hơn M;
      * Nếu chọn thêm món hàng thứ n thì tổng khối lượng được chọn trong (n-1) món hàng không lớn hơn (M-A[n]);
      * Suy ra bài toán có 2 tham số: số món hàng và khối lượng giới hạn.

# Lập công thức đệ quy

* + - * Gọi F(i, v) là tổng giá trị lớn nhất của các gói hàng được chọn trong i gói hàng sao cho tổng khối lượng không lớn hơn v .
      * Trường hợp A[i] > v: F(i, v) = F(i -1, v);
      * Trường hợp A[i] <= v:
  + Nếu gói hàng thứ i không được chọn thì: F(i, v) = F(i -1, v)
  + Nếu gói hàng thứ i được chọn thì: F(i, v) = F(i -1, v – A[i]) + C[i]

 F(i, v) = Max{ F(i -1, v); F(i -1, v – A[i]) + C[i]) };

Bài toán nhỏ nhất ứng với i = 0 ta có: F(0, v) = 0.

* + 1. Chương trình minh họa

#include <bits/stdc++.h> #define N 1000

#define M 1000 using namespace std; int dp[N][M];

int w[N],v[N],c[N]; int n,m;

long long sumv=0; int main()

{

ios\_base::sync\_with\_stdio(false); cin.tie(0); cout.tie(0);

freopen("CHONDO.INP", "r",stdin);

freopen("CHONDO.OUT", "w", stdout); cin>>n>>m;

for (int i=1; i<=n; i++) cin>>w[i]>>v[i];

for (int i=1; i<=n; i++) for (int j=1;j<=m;j++)

if (w[i]>j) dp[i][j]=dp[i-1][j];

else dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+v[i]); cout<<dp[n][m]<< "\n";

for (int i=n,j=m; i>0; i--)

if (j>=w[i] && dp[i][j]==dp[i-1][j-w[i]]+v[i])

{

c[i]=1;

sumv+=w[i]; j-=w[i];

}

// for (int i=1; i<=n; i++)

// if (c[i]==1) cout<<i<< " "; cout<<sumv;

return 0;

}

# Bài tập 6. Atcoder Educational DP Contest I - Coins (Mức độ khá)

*<https://oj.vnoi.info/problem/atcoder_dp_i>*

* + 1. Đề bài

Cho 𝑁 là một số nguyên dương lẻ. Có tất cả 𝑁 đồng xu được đánh số 1,2,...,𝑁. Xác suất để đồng xu thứ 𝑖 (1≤𝑖≤𝑁) lật mặt ngửa khi được tung lên là 𝑝𝑖 và xác suất để nó lật mặt sấp khi được tung lên là 1−𝑝𝑖.

Taro sẽ tung hết tất cả 𝑁 đồng xu. *Yêu cầu:* Tìm xác suất để có số mặt ngửa nhiều hơn số mặt sấp.

## Dữ liệu vào:

* Dòng đầu tiên chứa số nguyên dương lẻ 𝑁 (1≤𝑁≤2999)
* Dòng tiếp theo chứa 𝑁 số thực 𝑝𝑖 (0<𝑝𝑖<1), mỗi số có hai chữ số thập phân.

***Kết quả ra:*** In ra xác suất để có số mặt ngửa nhiều hơn số mặt sấp. Kết quả in ra được coi là chính xác nếu sai số tuyệt đối với đáp án không vượt quá 10−9.

## Ví dụ:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 3  0.30 0.60 0.80 | 0.612 |
| 1  0.50 | 0.5 |
| 5  0.42 0.01 0.42 0.99 0.42 | 0.3821815872 |

***Giải thích:***

+ Với ví dụ 1:

Xác suất để có số mặt ngửa nhiều hơn số mặt sấp bao gồm các trường hợp:

* Xác suất để có *(Ngửa, Ngửa, Ngửa)* là 0.3×0.6×0.8=0.144;
* Xác suất để có *(Sấp, Ngửa, Ngửa)* là 0.7×0.6×0.8=0.336;
* Xác suất để có *(Ngửa, Sấp, Ngửa)* là 0.3×0.4×0.8=0.096;
* Xác suất để có *(Ngửa, Ngửa, Sấp)* là 0.3×0.6×0.2=0.036. Do đó, xác suất để có số mặt ngửa nhiều hơn số mặt sấp là:

0.144 + 0.336 + 0.096 + 0.036 = 0.612.

+ Với ví dụ 2: Kết quả in ra ví dụ như 0.500, 0.500000001 và 0.499999999 đều được coi là chính xác.

* + 1. Phân tích bài toán

Gọi 𝑑𝑝[𝑖][𝑗] là xác suất để có 𝑗 mặt ngửa sau khi tung 𝑖 đồng xu đầu tiên. Xác suất này có thể được tính bằng cách lấy tổng xác suất của hai trường hợp:

* Đồng xu thứ 𝑖 là mặt ngửa, và 𝑖−1 đồng xu trước đó có 𝑗−1 mặt ngửa. Xác suất của trường hợp này là 𝑑𝑝[𝑖−1][𝑗−1]⋅𝑝[𝑖]
* Đồng xu thứ 𝑖 là mặt sấp, và 𝑖−1 đồng xu trước đó có 𝑗 mặt ngửa. Xác suất của trường hợp này là 𝑑𝑝[𝑖−1][𝑗]⋅(1−𝑝[𝑖])

Do đó, công thức quy hoạch động là:

𝑑𝑝[𝑖][𝑗]=𝑑𝑝[𝑖−1][𝑗−1]⋅𝑝[𝑖]+𝑑𝑝[𝑖−1][𝑗]⋅(1−𝑝[𝑖]) Độ phức tạp: *O**N* 2 

* + 1. Chương trình minh họa

#include<bits/stdc++.h> using namespace std;

long double dp[3001][3001]; int main() {

cin.tie(0)->sync\_with\_stdio(0); int n; cin >> n;

vector<long double> p(n + 1); for(int i = 1; i <= n; ++i) cin >> p[i]; int leastHeads = n / 2 + 1;

for (int i = 0; i <= n; ++i) dp[i][0] = 1; for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= leastHeads; ++j) {

dp[i][j] = dp[i - 1][j - 1] \* p[i] + dp[i - 1][j] \* (1 - p[i]);

}

}

cout << fixed << setprecision(10) << dp[n][leastHeads] << endl;

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/atcoder_dp_i>*

# Bài tập 7. Atcoder Educational DP Contest Q - Flowers (Mức độ khá)

*<https://oj.vnoi.info/problem/atcoder_dp_q>*

* + 1. Đề bài

Có 𝑁 bông hoa được xếp thành một hàng trong vườn hoa của Taro. Với mỗi 𝑖(1≤𝑖≤𝑁), chiều cao và vẻ đẹp của bông hoa thứ 𝑖 từ trái sang phải lần lượt là ℎ𝑖 và 𝑎𝑖. Trong vườn hoa, chiều cao của các bông hoa ℎ1,ℎ2,…,ℎ𝑁 phân biệt đôi một với nhau.

Taro đang nhổ một số bông hoa đi để những bông hoa còn lại trong vườn thỏa mãn điều kiện sau đây:

* Chiều cao của các bông hoa còn lại tăng dần từ trái sang phải.

Taro muốn vườn hoa của mình là đẹp nhất có thể (dĩ nhiên, ai mà chẳng muốn như vậy), vậy nên Taro muốn bỏ đi các bông hoa sao cho tổng vẻ đẹp của những bông hoa còn lại là lớn nhất. Tuy nhiên, Taro đang bận làm bài tập nên đã nhờ bạn

làm giúp việc này. Hãy giúp Taro và in ra tổng vẻ đẹp lớn nhất có thể của các bông hoa còn lại.

## Dữ liệu vào:

* Dòng thứ nhất chứa một số nguyên 𝑁(1≤𝑁≤2×105).
* Dòng thứ hai chứa 𝑁 số nguyên ℎ𝑖(1≤ℎ𝑖≤𝑁) phân biệt đôi một.
* Dòng thứ ba chứa 𝑁 số nguyên 𝑎𝑖(1≤𝑎𝑖≤109).

***Kết quả ra***: In ra tổng vẻ đẹp lớn nhất có thể của những bông hoa còn lại.

***Ví dụ***:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 4  3 1 4 2  10 20 30 40 | 60 |
| 1  1  10 | 10 |
| 5  1 2 3 4 5  1000000000 1000000000 1000000000 1000000000 1000000000 | 5000000000 |
| 9  4 2 5 8 3 6 1 7 9  6 8 8 4 6 3 5 7 5 | 31 |

# Giải thích

+ Đối với ví dụ 1: Tổng vẻ đẹp lớn nhất của vườn hoa có thể là 60, bằng cách giữ lại các bông hoa thứ hai và thứ tư từ trái qua phải.

+ Đối với ví dụ 2: Vườn hoa đã thỏa mãn điều kiện đề bài từ ban đầu.

+ Đối với ví dụ 4: Tổng vẻ đẹp lớn nhất của vườn hoa có thể là 31, bằng cách giữ lại các bông hoa thứ hai, thứ ba, thứ sáu, thứ tám và thứ chín từ trái qua phải.

* + 1. Phân tích bài toán

Gọi 𝑑𝑝[𝑖] là tổng giá trị lớn nhất của dãy bông hoa kết thúc tại 𝑖 và thỏa mãn điều kiện đề bài.

Khi đó, ta có: 𝑑𝑝[𝑖]=𝑚𝑎𝑥(𝑑𝑝[𝑗])+𝑎𝑖 với 𝑗<𝑖 và ℎ𝑗<ℎ𝑖.

Tuy nhiên, thử hết tất cả các giá trị của 𝑗 sẽ dẫn đến chương trình có độ phức tạp 𝑂(𝑁2), và dẫn đến TLE.

Chúng ta có thể sử dụng một cấu trúc dữ liệu để tìm giá trị của 𝑑𝑝[𝑖] nhanh hơn. Nhận thấy rằng khi đang tính 𝑑𝑝[𝑖], chúng ta đang tìm giá trị 𝑑𝑝[𝑗] lớn nhất mà ℎ𝑗<ℎ𝑖 và cộng vào 𝑎𝑖.

Nói cách khác, ta đang tìm 𝑑𝑝[𝑗] lớn nhất mà ℎ𝑗∈[1,ℎ𝑖−1], giá trị này có thể được tìm bằng một cấu trúc dữ liệu như [cây IT](https://vnoi.info/wiki/algo/data-structures/segment-tree-extend.md) hay [cây BIT](https://vnoi.info/wiki/algo/data-structures/fenwick.md) trong 𝑂(log𝑁).

Độ phức tạp: 𝑂(𝑁log𝑁)

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std; const int MAXN = 2e5; int n;

long long bit[MAXN + 1];

void update(int i, long long x) { for ( ; i <= n; i += i & -i)

bit[i] = max(bit[i], x);

}

long long query(int i) { long long ans = 0;

for ( ; i > 0; i -= i & -i) ans = max(ans, bit[i]);

return ans;

}

int main() {

cin.tie(0)->sync\_with\_stdio(0); cin >> n;

vector<int> h(n), a(n);

for (int i = 0; i < n; ++i) cin >> h[i]; for (int i = 0; i < n; ++i) cin >> a[i]; for (int i = 0; i < n; ++i) {

update(h[i], query(h[i] - 1) + a[i]);

}

cout << query(n) << '\n';

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/atcoder_dp_q>*

# Bài tập 8. Dãy chữ số (Mức độ khá)

* + 1. Đề bài

Cho 2 dãy chữ số a, b chỉ gồm các kí tự số ‘0’ … ‘9’. Dãy a được gọi là con của dãy b nếu a là một đoạn các kí tự số liên tiếp trong b. Ví dụ: ‘121’ là dãy con của dãy ‘22121’, nhưng không là dãy con của dã ‘112211’.

Dãy c gọi là dãy con chung của hai dãy a và b nếu c vừa là dãy con của a, vừa là dãy con của b.

**Yêu cầu:** Cho hai dãy chữ số a, b. Tìm độ dài của dãy con chung ngắn nhất c của a và b với điều kiện c chỉ xuất hiện đúng một lần trong a và đúng một lần trong b. **Dữ liệu vào:** Đọc từ tệp DAYCS.INP gồm:

* Dòng đầu ghi dãy chữ số a.
* Dòng thứ hai ghi dãy chữ số b.

Mỗi dãy con không quá 5000 kí tự chỉ gồm các kí tự số ‘0’…’9’.

**Output**: Ghi ra tệp DAYCS.OUT là độ dài của dãy con chung ngắn nhất c. Nếu không tồn tại c ghi – 1.

***Ví dụ:***

|  |  |
| --- | --- |
| **DAYCS.INP** | **DAYCS.OUT** |
| 1**231**23  312**231**2 | 3 |

# Giới hạn:

* Subtask 1: 25% số test ứng với 25% số điểm có độ dài mỗi dãy a, b không quá 10 kí tự.
* Subtask 2: 50% số test ứng với 50% số điểm có độ dài mỗi dãy a, b không quá 500 kí tự.
* Subtask 3: 25% số test ứng với 25% số điểm có độ dài mỗi dãy a, b không quá 5000 kí tự.
  + 1. Phân tích bài toán

Áp dụng tư tưởng quy hoạch động Bước 1:

* Tính fa[i] = độ dài đoạn con dài nhất kết thúc ở a[i] có lặp lại trong xâu a
* Tương tự tính fb[i] độ dài đoạn con dài nhất kết thúc ở b[i] có lặp lại trong xâu b

Bước 2:

* Tính q[i,j] = độ dài đoạn con chung dài nhất của 2 xâu a, b kết thúc ở a[i], b[j].
* Tại mỗi cặp vị trí (i, j), ta có q[i,j]=k thì a[i-k+1…i] và b[j-k+1…j] là đoạn con chung.
* fa[i] là đoạn lặp dài nhất kết thúc ở a[i] thì fa[i] + 1 sẽ chỉ xuất hiện 1 lần, tương tự fb[i]+1 cũng chỉ xuất hiện 1 lần trong xâu b.
* Đặt x = max(fa[i],fb[j]) + 1, nếu x < k thì đoạn a[i-x+1…x] và b[j-x+1…j] sẽ là đoạn chung chỉ xuất hiện 1 lần trong mỗi xâu a, b. Ta cập nhật với res để tìm đoạn ngắn nhất.
  + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std; string a,b;

int n, m,fa[100005],fb[100005]; int f[5005][5005],q[5005][5005];

int x, y,vc=1e9; int res = 1; void nhap()

{

getline(cin,a); getline(cin,b); m=a.size();

n=b.size();

a=' '+a;

b=' '+b;

}

void tinh\_fa()

{

for (int i=1;i<=m;i++)

{

for (int j=1;j<=m;j++)

{

if(i!=j&&a[i]==a[j]) f[i][j]=f[i-1][j-1]+1;

else f[i][j]=0; fa[i]=max(fa[i],f[i][j]);

fa[j]=max(fa[j],f[i][j]);

}

}

}

void tinh\_fb()

{

for (int i=1;i<=n;i++)

{

for (int j=1;j<=n;j++)

{

if(i!=j&&b[i]==b[j]) f[i][j]=f[i-1][j-1]+1;

else f[i][j]=0; fb[i]=max(fb[i],f[i][j]);

fb[j]=max(fb[j],f[i][j]);

}

}

}

void tim\_xau\_chung()

{

res=vc; int ir,jr;

for(int i=1;i<=m;i++)

{

for(int j=1;j<=n;j++)

{

if(a[i]==b[j]) q[i][j]=q[i-1][j-1]+1; else q[i][j]=0;

if(max(fa[i],fb[j])+1<=q[i][j])

{

//res=min(res,max(fa[i],fb[j])+1); if(q[i][j]<res)

{

res=q[i][j];

// res=min(res,q[i][j]); ir=i;jr=j;

}

}

}

}

if(res==vc) cout<<-1; else cout<<res;

// else cout<<res<<" "<<ir<<" "<<jr<<" "<<fa[ir]<<" "<<fb[jr];

}

void giai()

{

tinh\_fa(); tinh\_fb(); tim\_xau\_chung();

}

int main(){ ios\_base::sync\_with\_stdio(false); cin.tie(0); cout.tie(0);

freopen("daycs.inp", "r", stdin);

freopen("daycs.out", "w", stdout); nhap();

giai();

}

# Bài tập 8. MUA NHÀ (Mức độ khá)

* + 1. Đề bài

Để thu hút khách hàng mua vé sổ số của mình ngày một nhiều hơn công ty sổ số XYZ ngoài việc hàng ngày quay số và trao giải cho khách hàng công ty còn trao giải Độc đắc cho khách hàng sau mỗi chu kỳ *n* ngày.

Để được giải Độc đắc thì khách hàng phải có đủ *n* vé số của *n* ngày, vé số thứ

*i* phải có giá trị bằng *ai.*

Mới đây thầy NHM là người duy nhất nhận được giải Độc đắc và đã nhận được số tiền rất rất lớn. Thầy quyết định dùng tiền của mình mới nhận được để đầu cơ nhà, đất.

Dọc theo tuyến phố mới là dãy nhà đang được giao bán, các ngôi nhà đánh số từ 1, 2, … Thầy dự định dùng dãy số *a* may mắn ở trên để tạo ra tất cả các dãy con, nếu tổng của dãy con nào đó là một số nguyên tố thì thầy sẽ mua ngôi nhà được đánh số bằng tổng đó (Do ảnh hưởng của việc dạy đội tuyển trong nhiều năm).

Tuy nhiên vợ của thầy lại lại không muốn đầu tư vào đất. Cuối cùng họ đã thống nhất như sau: “Vợ của thầy ấy được phép chọn bỏ đi một phần tử nào đó trong dãy *a*, khi đó thầy NMH chỉ được phép sử dụng *n-1* phần tử còn lại”.

Bạn hãy lập trình giúp vợ thầy NMH chọn ra một phần tử để sao cho số ngôi nhà mà thầy NMH sẽ mua là ít nhất có thể.

**Dữ liệu:** Vào từ tệp văn bản muanha.inp.

- Dòng đầu chứa hai số nguyên , 𝑘 (1 ≤ 𝑛 ≤ 100; 0 ≤ 𝑘 ≤ 1).

- Dòng thứ hai chứa 𝑛 số nguyên dương 𝑎1, 𝑎2, . . , 𝑎𝑛 (1 ≤ 𝑎𝑖 ≤ 103).

**Kết quả:** Ghi ra tệp văn bản muanha.out một số nguyên duy nhất là số nhà ít nhất mà thầy NMH có thể mua.

# Ví dụ:

|  |  |  |
| --- | --- | --- |
| **muanha.inp** | **muanha.out** | **Giải thích** |
| 3 0  2 6 3 | 4 | Các nhà mua được có các số:2, 3, 5, 11 |
| 3 1  2 6 5 | 1 | Bỏ a3=5; Khi đó chỉ mua được nhà có số: 2 |

**Subtasks:**

* Subtask 1 (30%): n ≤ 20; 𝑘 = 0;
* Subtask 2 (30%): 𝑛 ≤ 20; 𝑘 = 1.
* Subtask 3 (20%): 𝑛 ≤ 100; 𝑘 = 0; 1 ≤ 𝑎𝑖 ≤ 102.
* Subtask 4 (20%): 𝑛 ≤ 100; 𝑘 = 1; 1 ≤ 𝑎𝑖 ≤ 102.
  + 1. Phân tích bài toán
       - Subtask 1, 2 (30% + 30%): Dùng đệ qui tạo dãy nhị phân.
       - Subtask 3 (20%): Dùng qui hoạch động dãy con có tổng bằng S.
       - Subtask 4 (20%): Duyệt loại bỏ từng phần tử rồi áp dụng subtask 3.
    2. Chương trình minh họa #include <bits/stdc++.h>

#define REP(i,n) for(int i=0;i<(n);++i)

#define FOR(i,a,b) for(int i=(a);i<=(b);++i) #define FORD(i,a,b) for(int i=(a);i>=(b);--i)

#define FOREACH(i,c) for( typeof((c).begin()) i=(c).begin();i!=(c).end();++i) #define builtin\_popcount builtin\_popcountll

#define BIT(x,i) (((x)>>(i))&1)

#define sz(x) ((int)(x).size()) #define MASK(x) (1<<(x)) #define ll long long

#define pb push\_back #define mp make\_pair #define se second #define fi first

#define all(c) ((c).begin()), ((c).end()) #define LOCAL\_DEBUG

using namespace std; const int inf=1e9; const ll INF=1e18; int numt;

typedef pair <int, int> ii;

typedef vector <int> vi;

template <class T> bool minimize(T &x,T y){if(x>y){x=y;return true;}return false;}

template <class T> bool maximize(T &x,T y){if(x<y){x=y;return true;}return false;}

void setIO(string name = ""){ cin.tie(0)->sync\_with\_stdio(0); if (sz(name)) {

freopen((name + ".inp").c\_str(), "r", stdin);

freopen((name + ".out").c\_str(), "w", stdout);

}

}

int n, k;

const int N = 101, LIM = 10005; bool prime[LIM];

bool f[LIM]; int a[N];

void solve2() {

int ans = inf;

for (int del = 1; del <= n; del++) { swap(a[del], a[n]);

memset(f, 0, sizeof f); f[0] = true;

for (int i = 0; i < n - 1; i++)

for (int sum = LIM - 1; sum >= 0; sum--) if (f[sum]) { if (sum + a[i + 1] < LIM) f[sum + a[i + 1]] = true; f[sum] = true;

}

int count = 0;

for (int i = 0; i < LIM; i++)

if (prime[i] && f[i]) count++; minimize(ans, count); swap(a[del], a[n]);

}

cout << ans;

}

void solve1() { f[0] = true;

for (int i = 0; i < n; i++)

for (int sum = LIM - 1; sum >= 0; sum--) if (f[sum]) { if (sum + a[i + 1] < LIM) f[sum + a[i + 1]] = true; f[sum] = true;

}

int count = 0;

for (int i = 0; i < LIM; i++)

if (prime[i] && f[i]) count++; cout << count;

}

void sieve() {

for (int i = 2; i < LIM; i++) prime[i] = true; for (int i = 2; i \* i < LIM; i++)

if (prime[i])

for (int j = i \* i; j < N; j += i) prime[j] = false;

}

int main()

{

setIO("muanha"); cin >> n >> k;

FOR(i, 1, n) cin >> a[i]; sieve();

if (k == 0) solve1(); else solve2();

}

# Bài tập 9. KINH DOANH (Mức độ khá)

*<https://oj.vnoi.info/problem/bedao_m22_e>*

* + 1. Đề bài

Gia đình nhà Nam là một doanh nghiệp lớn, bản thân Nam vừa tốt nghiệp ngành Quản trị kinh doanh nên Nam được gia đình giao cho quản lí chuỗi cửa hàng tiện lợi Thành Đạt. Áp dụng công nghệ số, Nam dùng phần mềm quản lí kinh doanh cập nhật lợi nhuận theo từng ngày trong 𝒏 ngày, trong đó ngày thứ 𝒊 đạt lợi nhuận là 𝒂𝒊.

Nhân kỉ niệm ngày thành lập công ty, Nam muốn minh chứng việc kinh doanh ngày càng phát triển bằng cách chọn ra một ngày hoặc một số ngày trong 𝒏 ngày trên thỏa mãn lợi nhuận ngày sau lớn hơn lợi nhuận ngày trước và tổng lợi nhuận các ngày được chọn là lớn nhất. Các bạn hãy giúp Nam giải quyết vấn đề trên.

**Dữ liệu:** Vào từ tệp văn bản **KINHDOANH.INP** gồm:

* + - * Dòng đầu tiên gồm số nguyên dương 𝑛 (1 ≤ 𝑛 ≤ 105);
      * Dòng thứ hai gồm 𝑛 số nguyên không âm 𝑎1, 𝑎2, . . , 𝑎𝑛(0 ≤ 𝑎𝑖 ≤ 109𝑣ớ𝑖 1 ≤

𝑖 ≤ 𝑛).

**Kết quả:** Ghi ra tệp văn bản **KINHDOANH.OUT** một số nguyên duy nhất là tổng lợi nhuận lớn nhất.

**Ví dụ:**

|  |  |  |
| --- | --- | --- |
| **KINHDOANH.INP** | **KINHDOANH.OUT** | **GIẢI THÍCH** |
| 6  1 7 2 7 8 5 | 18 | Chọn các ngày:1, 3, 4, 5 tương ứng với dãy con:  1, 2, 7, 8 |

**Ràng buộc:**

* + - * *Có* 30% *số test tương ứng với* 30% *số điểm của bài có* 𝑛 ≤ 20*;*
      * *Có* 40% *số test tương ứng với* 40% *số điểm của bài có* 20 < 𝑛 ≤ 1000*;*
      * *Có* 30% *số test tương ứng với* 30% *số điểm của bài có* 1000 < 𝑛 ≤ 105*.*
    1. Phân tích bài toán

# Subtaks 1:

Duyệt đệ quy dãy nhị phân độ dài *n* với *x[i]=1* nếu giữ lại phần tử thứ *i.*

=> Độ phức tạp là: *O(2n)*

# Subtask 2:

* Quy hoạch động dãy con tăng dần:
* Gọi *F[i]* là tổng lớn nhất của dãy con tăng dần khi xét đến phần tử thứ *i*

trong mảng *a* và có lấy phần tử *a[i]* và dãy con.

* Công thức QHĐ: *F[i]= max (F[j]) + a[i] với j=1..i-1 và a[j]<a[i].*

=> Độ phức tạp *O(n2)*

# Subtask 3:

* Cải tiến QH của subtask 2.
* Với mỗi phần tử thứ *i* ta cần tìm giá trị lớn nhất của mảng F mà kết thúc bởi một dãy con có phần tử cuối cùng < a[i]. Việc này ta có thể xử dụng cây BIT để tìm kiếm. Tuy nhiên do |ai| ≤ 109 nên ta phải dùng thêm cấu trúc map để rời rạc hóa giá trị của các phần tử.

=> Độ phức tạp: **O(nlogn)**

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std;

using ll = long long; const int N = 1e5 + 5; int n;

int a[N];

ll s[N], f[N], tr[N << 2], st[N];

void build(int id, int l, int r)

{

if(l == r)

{

st[l] = id; return;

}

int m = l + r >> 1; build(id << 1, l, m);

build(id << 1 | 1, m + 1, r);

}

void upd(int x, ll val)

{

int id = st[x];

tr[id] = max(tr[id], val); while(id)

{

id /= 2;

tr[id] = max(tr[id << 1], tr[id << 1 | 1]);

}

}

ll get(int id, int l, int r, int u, int v)

{

if(r < u || l > v) return 0;

if(r <= v && l >= u) return tr[id]; int m = l + r >> 1;

return max(get(id << 1, l, m, u, v), get(id << 1 | 1, m + 1, r, u, v));

}

vector<ll> ve; signed main()

{

cin.tie(0) -> ios\_base::sync\_with\_stdio(false); freopen("KINHDOANH.INP", "r", stdin); freopen("KINHDOANH.OUT", "w", stdout);

cin >> n;

for(int i = 1; i <= n; i++) cin >> a[i]; build(1, 1, n);

ve.resize(n);

for(int i = 1; i <= n; i++)

{

ve[i - 1] = a[i];

s[i] = a[i];

}

sort(ve.begin(), ve.end()); ve.resize(unique(ve.begin(), ve.end()) - ve.begin()); for(int i = 1; i <= n; i++)

a[i] = lower\_bound(ve.begin(), ve.end(), a[i]) - ve.begin() + 1; ll ans = 0;

for(int i = 1; i <= n; i++)

{

f[i] = get(1, 1, n, 1, a[i] - 1) + s[i]; ans = max(ans, f[i]);

upd(a[i], f[i]);

}

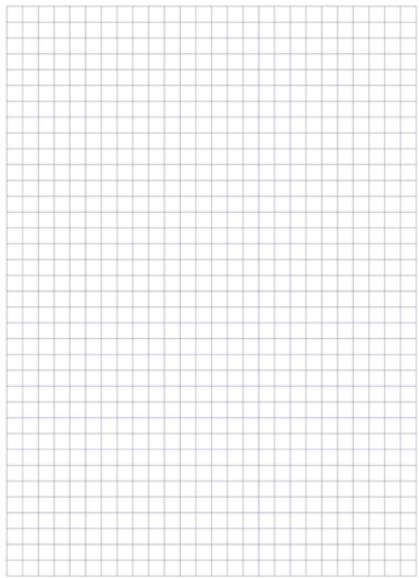
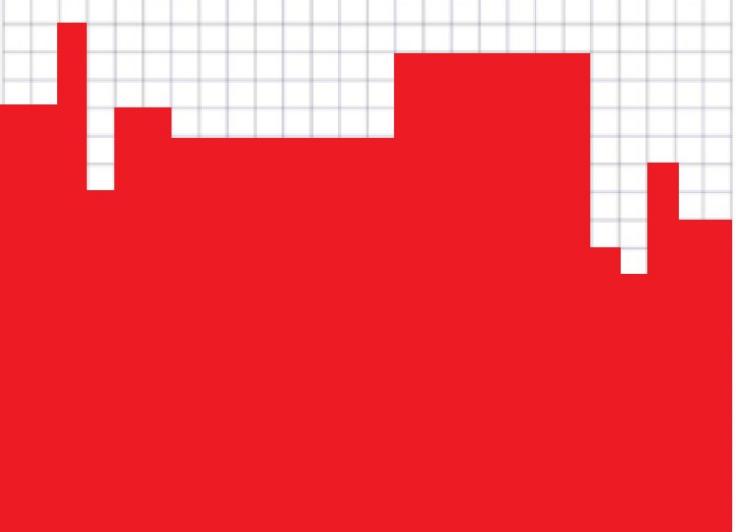
cout << ans; return 0;

}

# Bài tập 10. VẼ BIỂU ĐỒ (Mức độ khó)

* + 1. Đề bài

Huy là một kỹ sư tài năng, hiện anh đang thực hiện một nghiên cứu mới của mình và cần phải thực hiện vẽ biểu đồ cột, biểu đồ này của Huy rất đặc biệt bởi các cột của chúng nằm liền sát nhau và không có khoảng cách. Vì vậy Huy quyết định sẽ vẽ biểu đồ trên tấm giấy kẻ ô (Caro Paper) kích thước 109 × 𝑛 của mình.

Ví dụ về Caro Paper và Biểu đồ cột mà Huy cần vẽ

Cụ thể hơn, biểu đồ mà Huy cần vẽ có 𝑛 cột, cột thứ 𝑖 có độ cao là ℎ𝑖; người giỏi thường lười, Huy cũng không phải là ngoại lệ, anh muốn vẽ nên biểu đồ thật nhanh, vì vậy Huy sẽ chỉ thực hiện liên tục thao tác:

- Chọn một số ô liên tiếp trên một hàng và tô màu các ô đó

Vì dành quá nhiều thời gian cho việc nghiên cứu, Huy đã làm bạn nữ (giấu tên) cảm thấy buồn; cô quyết định nhân lúc Huy đi vắng sẽ sửa chiều cao của một số cột để anh làm việc thật nhanh và có thời gian quan tâm mình. Để không ảnh hưởng quá nhiều tới kết quả nghiên cứu, bạn gái ấy sẽ chỉ sửa độ cao của không quá 𝑘 cột.

**Yêu cầu:** Bạn hãy giúp bạn nữ (giấu tên) sửa lại chiều cao của các cột sao cho số thao tác mà Huy thực hiện là ít nhất nhé.

**Dữ liệu:** Vào từ file văn bản RESEARCH.INP:

* Dòng đầu tiên chứa hai số nguyên 𝑛, 𝑘 (0 ≤ 𝑘 ≤ 𝑛 ≤ 2000) là số cột của biểu đồ và số cột tối đa bạn nữ sẽ sửa
* Dòng thứ hai chứa 𝑛 số nguyên không âm ℎ1, ℎ2, … , ℎ𝑛 (ℎ𝑖 ≤ 109) là độ cao ban đầu của 𝑛 cột

**Kết quả:** Xuất ra file văn bản RESEARCH.OUT:

* Một số nguyên duy nhất là số thao tác nhất Huy sẽ phải thực hiện trong cách sửa tối ưu tìm được.

# Ràng buộc:

* Có 25% số điểm của bài thỏa mãn 𝑘 = 0
* Có 25% số điểm khác với ℎ1 ≤ ℎ2 ≤ ⋯ ≤ ℎ𝑛
* Có 25% số điểm khác có 𝑛 ≤ 500
* Còn lại không có điều kiện gì thêm.

# Ví dụ:

|  |  |  |
| --- | --- | --- |
| **RESEARCH.INP** | **RESEARCH.OUT** | **Giải thích:** |
| 4 1  1 2 3 4 | 3 | Đổi ℎ4 thành 3, cần 3 lần tô màu:   * Tô hàng 1 của tất cả cá cột * Tô hàng 2 từ cột 2 tới cột 4 * Tô hàng 3 từ cột 3 tới cột 4 |
| 6 2  1 2 1 9 6 8 | 7 | Đổi ℎ3 thành 2 và ℎ4 thành 6, ta mất 7 lần tô màu |

* + 1. Phân tích bài toán

Khi 𝑘 = 0, có thể chứng minh đáp án là ℎ1 + max(0, ℎ2 − ℎ1) + max(0, ℎ3 − ℎ2) +

⋯ + max(0, ℎ𝑛 − ℎ𝑛−1):

* + - * Nếu ℎ𝑖 ≤ ℎ𝑖−1: Mọi hàng của cột 𝑖 sẽ có thể được tô với hàng tương ứng của cột 𝑖 − 1
      * Nếu ℎ𝑖 > ℎ𝑖−1: ℎ𝑖 − ℎ𝑖−1 hàng từ ℎ𝑖−1 + 1 → ℎ𝑖 của cột 𝑖 bắt buộc phải có các nét vẽ riêng.

⇒ Ta sử dụng quy hoạch động để giải quyết bài toán:

Gọi 𝑑𝑝[𝑖][𝑡] là số lần phải tô ít nhất khi xét các cột 1,2, … , 𝑖 (cột thứ 𝑖 không thay đổi độ cao) và số cột thay đổi là 𝑡.

Giả sử 𝑖 và 𝑗 là hai cột liên tiếp **không bị đổi** độ cao, rõ ràng trong đáp án tối ưu các cột ℎ𝑖+1, ℎ𝑖+2, … , ℎ𝑗−1 sẽ đổi độ cao thành ℎ𝑖 hoặc ℎ𝑗.

⇒ Ta có công thức truy hồi:

𝑑𝑝[𝑖][𝑡] = min

0≤𝑗<𝑖

(𝑑𝑝[𝑗][𝑡 − (𝑖 − 𝑗 − 1)] + 𝑚𝑎𝑥(0, ℎ𝑖 − ℎ𝑗))

𝑗≥𝑖−𝑡−1

Coi cột 𝑛 + 1 là cột phải cùng với độ cao ℎ𝑛+1 = 0. Đáp án lúc này là

min

0≤𝑡≤𝑘

𝑑𝑝[𝑛 + 1][𝑡].

Để giải được subtask cuối, ta cần để ý thấy 𝑑𝑝[𝑖][𝑡] được cập nhật từ:

- 𝑑𝑝[𝑖 − 1][𝑡] + 𝑚𝑎𝑥(0, ℎ[𝑖] − ℎ[𝑖 − 1])

- 𝑑𝑝[𝑖 − 2][𝑡 − 1] + 𝑚𝑎𝑥(0, ℎ[𝑖] − ℎ[𝑖 − 2])

- 𝑑𝑝[𝑖 − 3][𝑡 − 2] + 𝑚𝑎𝑥(0, ℎ[𝑖] − ℎ[𝑖 − 3])

- …

- 𝑑𝑝[𝑖 − 𝑡 − 1][0] + …

⇒ 𝑑𝑝[𝑖][𝑡] sẽ được cập nhật từ các trạng thái 𝑑𝑝[𝑥][𝑦] 𝑚à 𝑥 − 𝑦 = 𝑖 − 𝑡 − 1 (một đường chéo trên bảng)

Vậy ta sẽ duy trì hai hàm mục tiêu phụ như sau:

𝑓(𝑑, 𝑣) = min (𝑑𝑝[𝑖][𝑡])

𝑖−𝑡=𝑑

ℎ𝑖 ≥ 𝑣

𝑔(𝑑, 𝑣) = min (𝑑𝑝[𝑖][𝑡] − ℎ )

𝑖

𝑖−𝑡=𝑑

ℎ𝑖 ≤ 𝑣

Khi đó công thức truy hồi trở thành:

𝑑𝑝[𝑖][𝑡] = min(𝑓(𝑖 − 𝑡 − 1, ℎ𝑖), 𝑔(𝑖 − 𝑡 − 1, ℎ𝑖) + ℎ𝑖)

{ 𝑓(𝑖 − 𝑡, 𝑣) = min(𝑓(𝑖 − 𝑡, 𝑣), 𝑑𝑝[𝑖][𝑡]) ∀𝑣 ≤ ℎ𝑖

𝑔(𝑖 − 𝑡, 𝑣) = min(𝑔(𝑖 − 𝑡, 𝑣), 𝑑𝑝[𝑖][𝑡] − ℎ𝑖) ∀𝑣 ≥ ℎ𝑖

Đến đây, nếu duy trì được hai hàm mục tiêu 𝑓 và 𝑔 thì lời giải hoàn tất, đây là bài toán quen thuộc sử dụng Nén số và Cấu trúc dữ liệu.

* + 1. Chương trình minh họa

#include<iostream> #include<vector> #include<algorithm> #pragma GCC target ("avx2")

#pragma GCC optimization ("O3") #pragma GCC optimization ("unroll-loops") #pragma GCC optimize("Ofast")

#pragma GCC target("avx,avx2,fma")

//#define int long long using namespace std; using ll = long long; const int maxN = 2e3 + 5;

const int mod = 998244353; const ll infty = 1e18;

int n,k;

ll dp[2005][2005];

vector<int> unq; int h[maxN]; struct Segtree

{

ll st[4\*maxN];

void Init()

{

fill(st,st + 4\*n + 5,infty);

}

void Update(int idx,int l,int r,int i,ll val)

{

if(i < l || i > r) return; if(l == r)

{

st[idx] = min(st[idx],val); return;

}

int mid = (l + r) / 2; Update(idx \* 2,l,mid,i,val);

Update(idx \* 2 + 1,mid+1,r,i,val); st[idx] = min(st[idx\*2],st[idx\*2+1]);

}

ll Get(int idx,int l,int r,int u,int v)

{

if(v < l || u > r) return infty; if(u <= l && r <= v)

return st[idx]; int mid = (l + r) / 2;

return min(Get(idx\*2,l,mid,u,v),Get(idx\*2+1,mid+1,r,u,v));

}

};

Segtree small[2005],big[2005]; inline int new\_val(int x)

{

return upper\_bound(unq.begin(),unq.end(),x) - unq.begin();

}

void Read()

{

cin >> n >> k;

for(int i = 1;i <= n;i++)

{

cin >> h[i]; unq.push\_back(h[i]);

}

for(int i = 0;i <= n;i++)

{

small[i].Init();

big[i].Init();

for(int j = 0;j <= k;j++)

{

dp[i][j] = 1e18;

}

}

unq.push\_back(0); sort(unq.begin(),unq.end());

unq.erase(unique(unq.begin(),unq.end()),unq.end()); dp[0][0] = 0;

small[0].Update(1,1,n+1,new\_val(0),0);

big[0].Update(1,1,n+1,new\_val(0),0); for(int i = 1;i <= n;i++)

{

dp[i][0] = dp[i-1][0] + max(0,h[i] - h[i-1]);

small[i].Update(1,1,n+1,new\_val(h[i]),dp[i][0]-h[i]);

big[i].Update(1,1,n+1,new\_val(h[i]),dp[i][0]);

}

for(int j = 1;j <= k;j++)

{

for(int i = 0;i <= n;i++)

{

//dp[i][j] = min(dp[i][j],dp[i][j-1]); if(i - 1 < j) continue;

int x = new\_val(h[i]);

dp[i][j] = min(small[i-j-1].Get(1,1,n+1,1,x) + h[i],dp[i][j]);

dp[i][j] = min(big[i-j-1].Get(1,1,n+1,x+1,n+1),dp[i][j]);

small[i-j].Update(1,1,n+1,x,dp[i][j]-h[i]);

big[i-j].Update(1,1,n+1,x,dp[i][j]);

}

}

ll res = 1e18;

for(int j = 0;j <= k;j++)

{

for(int i = 0;i <= n;i++)

{

if(j + (n - i) <= k)

{

res = min(res,dp[i][j]);

}

}

}

cout << res;

}

void Solve()

{

}

void Debug()

{

}

int32\_t main()

{

freopen("research.inp","r",stdin); freopen("research.out","w",stdout); ios\_base::sync\_with\_stdio(0); cin.tie(0);

// int subtask; int tt;

tt = 1;

//cin >> tt;

for(int i = 1; i <= tt; i++)

{

Read();

Solve();

Debug();

}

}

# Bài tập 11. Dãy con tăng dài nhất (Mức độ khó)

*<https://oj.vnoi.info/problem/fcc2023_lis>*

* + 1. Đề bài

Cho dãy *a* ,*a* ,...,*a* gồm n phần tử (với 1  *n*  109 ) được mã hóa thành mảng

1 2 *n*

*b*1,*b*2,...,*bn* gồm m phần tử như sau: Dãy b sẽ gồm các phần tử có dạng x.y với ý

nghĩa: Phần tử hiện đang có giá trị là x, y phần tử tiếp theo lập thành một dãy cấp số cộng với công sai là 1. Xét a ={1,2,3,2,9,4,5,6}. Ta có thể mã hóa mảng a thành mảng b = {1.3,2.1,9.1,4.3} như sau:

* + - * Tại vị trí i = 1, ta thấy từ i = 1 đến j = 3 lập thành một dãy cấp số cộng có công sai 1 với u0 = 1, gồm 3 phần tử → b ={1.3}.
      * Tại vị trí i = 4, ta thấy từ i = 4 đến j = 4 lập thành một dãy cấp số cộng có công sai 1 với u0 = 2, gồm 1 phần tử → b ={1.3,2.1}.
      * Tại vị trí i = 5, ta thấy từ i = 5 đến j = 5 lập thành một dãy cấp số cộng có công sai 1 với u0 = 9, gồm 1 phần tử → b ={1.3,2.1,9.1}
      * Tại vị trí i = 6, ta thấy từ i = 6 đến j = 8 lập thành một dãy cấp số cộng có công sai 1 với u0 = 4, gồm 3 phần tử → b ={1.3,2.1,9.1,4.3}.

Sau khi được mã hóa, mảng b sẽ gồm m phần tử (với 1  *m* 105 ). Từ dãy b

cho trước, tìm độ dài của dãy con tăng dần dài nhất của dãy a ban đầu khi chưa được mã hóa sang dãy b.

# Dữ liệu vào:

* Một số nguyên m, là số phần tử của mảng *b* ,*b* ,...,*b* 1  *m*  105 .

1 2

*m*

* m dòng tiếp theo, mỗi dòng gồm hai số nguyên dương *xi* và *yi*

1  *x*  106, 1  *y*  109, *x*  *y*  109 .

*i i i i*

* Dữ liệu đầu vào đảm bảo số lượng phần tử của dãy a ban đầu có không quá 109 phần tử.

**Kết quả ra:** Một số nguyên duy nhất là độ dài của dãy con tăng dần dài nhất của dãy a.

# Ví dụ:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |

|  |  |
| --- | --- |
| 3  1 3  4 2  2 7 | 8 |
| 4  1 3  2 1  9 1  4 3 | 6 |

**Ràng buộc**

* + Subtask 1 (5% số test): dãy a ban đầu có dạng *ai* 

*ai*1 1 (*i* :1  *i* 

*n*) .

* Subtask 2 (10% số test): 1 
* Subtask 3 (15% số test): 1 
* Subtask 4 (25% số test): 1 

*n* 103.

*n*  105.

*m*  103.

* Subtask 5 (45% số test): Không có ràng buộc gì thêm.
  + 1. Phân tích bài toán

*Tóm tắt đề bài:* Cho dãy b đã mã hóa, tìm LIS trên dãy a ban đầu.

## Ý tưởng và lời giải

* + - * **Subtask 1** (5%) Rõ ràng là đáp án sẽ là tổng yi. Việc này có thể thực hiện trong O(m).
      * **Subtask 2** (10%) Vì n tương đối nhỏ *n*  103  , nên ta có thể khởi tạo lại dãy a

ban đầu và thực hiện quy hoạch động tìm dãy con tăng dần lớn nhất (LIS) trên dãy a trong O(n2). Gọi dpi là dộ dài dãy con tăng dần lớn nhất kết thúc tại vị trí i. Xét các

vị trí *j*  *j*  *i*, nếu *Aj*  *Ai* , ta có thể cập nhật *dpi*  *max**dpi* , *dpj* 1.

* + - * **Subtask 3** (15%) Ở subtask này ta vẫn có đủ khả năng khởi tạo lại dãy a ban đầu. Tuy nhiên, với giới hạn *n* khá lớn ta thì ta phải nghĩ đến cách thực hiện QHĐ LIS một cách tối ưu hơn. Một trong số đấy có thể tìm hiểu ở đây: LIS- cp-algorithms

Lưu ý: Giới hạn giá trị phần ai có thể lớn *a*  109  nên đồng thời bạn cũng cần biết đến rời rạc hóa (dân gian hay gọi là nén giá trị). Độ phức tạp: *O* *NlogN* .

*i*

* + - * **Subtask 4** (25%) Đối với subtask 4, giới hạn n có thể rất lớn (n ≤ 109), ta sẽ tận dụng **Subtask 4** giới hạn m khá nhỏ *m*  103 .
        + Gọi dpi là độ dài LIS của dãy a ban đầu khi xét LIS có các phần tử cuối đều có các giá trị thuộc dãy cấp số cộng thứ i của b.
        + Với mỗi phần tử thứ i trong b, xét các vị trí *j*  *j*  *i* , có 2 trường hợp ảnh

hưởng đến kết quả *dpi* :

Nếu *xj*  *yj*  *xi* : Tức 2 dãy cấp số cộng j và i không giao nhau. Như vậy

ta có công thức: *dpi*  *max**dpi* , *dpj*  *yi*  1

Nếu *xi*  *xj*  *yj*  *xi*  *yi* : 2 dãy cấp số cộng j và i giao nhau, giá trị phần

tử lớn nhất của dãy j nhỏ hơn giá trị lớn nhất của dãy i:

*dpi*  *max**dpi* , *dpj*   *xi*  *yi*   *xj*  *y j*  2

– Nếu *xi*  *yi*  *xj*  *y j* hoặc *xi*  *yi*  *xj* : Ở trường hợp này, rõ ràng là các phần tử

cuối của LIS sẽ không thuộc hết vào dãy cấp số cộng thứ i của b, vì vậy, đáp án của trường hợp này luôn là 0. Như vậy, chúng ta không phải xét trường hợp này.

Độ phức tạp: O(m2) cho việc tính toán, sử dụng *O**m* bộ nhớ.

* + - * **Subtask 5** (45%) Nếu chú ý kỹ, bạn sẽ thấy các công thức ở subtask 4 đều có thể được tính toán nhanh hơn bằng việc sử dụng các cấu trúc dữ liệu như Fenwick Tree và Segment Tree. Áp dụng được các CTDL trên, ta có thể giảm độ phức tạp bài toán

từ *O**m*2  xuống còn *O**mlogm*,phù hợp với giới hạn thời gian của bài.

Độ phức tạp: *O**mlogm*

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std;

#define BIT(x, i) (1 & ((x) >> (i))) #define MASK(x) (1LL << (x)) #define OFF(x, i) ((x) & ~(1LL << (i)))

#define ON(x, i) ((1LL << (i)) | (x)) #define CNT(x) builtin\_popcountll(x) #define endl '\n'

#define F first

#define S second

#define all(v) (v).begin(), (v).end()

#define TIME (1.0 \* clock() / CLOCKS\_PER\_SEC) #define int long long

#define log2(x) 63 - builtin\_clzll(x)

#define FOR(i, l, r) for(int i = (l), \_r = (r); i <= \_r; ++i) #define FORD(i, l, r) for(int i = (l), \_r = (r); i >= \_r; --i) #define pii pair<int, int>

int const N = 1e5 + 5; int const INF = 1e18;

int m, a[(int)2e6 + 6], sum; pii p[N];

struct Fenwick\_Tree

{

int n; vector<int> bit;

Fenwick\_Tree(int \_n)

{

n = \_n;

bit.assign(n + 7, 0);

}

void Update(int i, int val)

{

if (i == 0) ++i; while (i <= n)

{

bit[i] = max(bit[i], val); i += i & -i;

}

}

int Get(int x)

{

int res = 0; while (x > 0)

{

res = max(res, bit[x]); x -= x & -x;

}

return res;

}

};

bool CheckS1()

{

int last = p[1].F + p[1].S - 1; FOR(i, 2, m)

{

if (last + 1 != p[i].F) return false; last = p[i].F + p[i].S - 1;

}

return true;

}

void Sub2()

{

int dem = 0;

FOR(i, 1, m) FOR(j, 1, p[i].S) a[++dem] = p[i].F + j - 1;

Fenwick\_Tree bit(2e6); FOR(i, 1, dem)

{

int cur = bit.Get(a[i] - 1) + 1; bit.Update(a[i], cur);

}

cout << bit.Get(2e6);

}

int dp[N]; // LIS length of the initial sequence when considering LIS ending at elements belonging to the i-th arithmetic progression of b

void Sub4()

{

dp[0] = 0, dp[1] = p[1].S; FOR(i, 2, m)

{

dp[i] = p[i].S;

FOR(j, 0, i - 1) if (p[j].F <= p[i].F)

{

if (p[j].F + p[j].S <= p[i].F) dp[i] = max(dp[i], dp[j] + p[i].S);

else if (p[i].F <= p[j].F + p[j].S && p[j].F + p[j].S <= p[i].F + p[i].S)

dp[i] = max(dp[i], dp[j] + p[i].F + p[i].S - p[j].F - p[j].S);

}

}

cout << \*max\_element(dp + 1, dp + 1 + m); exit(0);

}

vector<int> Nen; struct SegMentTree

{

int Tree[N << 2];

void Update(int u, int v, int val, int k = 1, int l = 1, int r = m)

{

if (u > r || v < l) return; if (l == r && l == u)

{

Tree[k] = max(Tree[k], val); return;

}

int mid = (l + r) >> 1;

Update(u, v, val, k << 1, l, mid); Update(u, v, val, k << 1 | 1, mid + 1, r);

Tree[k] = max(Tree[k << 1], Tree[k << 1 | 1]);

}

int Get(int u, int v, int k = 1, int l = 1, int r = m)

{

}

} IT;

if (u > r || v < l) return -INF;

if (u <= l && r <= v) return Tree[k]; int mid = (l + r) >> 1;

return max(Get(u, v, k << 1, l, mid), Get(u, v, k << 1 | 1, mid + 1, r));

void Sub5()

{

sort(all(Nen));

Nen.resize(unique(all(Nen)) - Nen.begin()); FOR(i, 1, 4 \* m) IT.Tree[i] = -INF;

Fenwick\_Tree bit(m); FOR(i, 1, m)

{

int l = lower\_bound(all(Nen), p[i].F) - Nen.begin() + 1;

int r = lower\_bound(all(Nen), p[i].F + p[i].S) - Nen.begin() + 1; dp[i] = p[i].S + bit.Get(l - 1);

dp[i] = max(dp[i], p[i].F + p[i].S + IT.Get(l, r)); bit.Update(r, dp[i]);

IT.Update(r, r, dp[i] - p[i].F - p[i].S);

}

cout << \*max\_element(dp + 1, dp + 1 + m);

}

signed main()

{

ios\_base::sync\_with\_stdio(false); cin.tie(NULL);

cout.tie(NULL); cin >> m; FOR(i, 1, m)

{

cin >> p[i].F >> p[i].S; sum += p[i].S;

Nen.push\_back(p[i].F + p[i].S);

}

if (sum <= 1e6) Sub2();

else if (CheckS1()) cout << sum; else Sub5();

cerr << endl << TIME << endl; return 0;

}

* + 1. Test trực tiếp trên link *<https://oj.vnoi.info/problem/fcc2023_lis>*

# Bài tập 11. Thi tốt nghiệp (Mức độ khó)

* + 1. Đề bài

Các bạn nhỏ của trường mầm non SuperKids vừa mới hoàn thành kì thi tốt nghiệp một cách xuất sắc! Giáo sư X muốn ước lượng độ chênh lệch điểm của các bạn trong lớp.

Lớp học có 𝑛 bạn, và sau khi thi, bạn thứ 𝑖 nghĩ ước lượng mình sẽ được 𝑎𝑖 điểm (số điểm có thể âm). Giáo sư X sẽ dựa vào các điểm này để tính toán độ chênh lệch điểm số như sau:

* + - 1. Thầy chia lớp làm một số các nhóm, mỗi nhóm sẽ gồm một số các bạn liên

tiếp nhau

*al* ,*al*1,...,*ar*

nào đó. Mỗi bạn sẽ thuộc đúng một nhóm.

* + - 1. Với mỗi nhóm, thầy sẽ chọn một bạn để đưa vào danh sách các học sinh làm bài tốt, và một bạn để đưa vào danh sách các học sinh làm bài không tốt (một bạn có thể thuộc cả hai nhóm học sinh này).
      2. Độ chênh lệch của một cách chia và chọn sẽ bằng tổng số điểm của các học sinh làm bài tốt, trừ cho tổng số điểm của các học sinh làm bài không tốt.
      3. Độ chênh lệch của lớp sẽ bằng độ chênh lệch tối đa của mọi cách chia nhóm và danh sách học sinh làm bài tốt, không tốt.

Trong những ngày tiếp theo, sẽ có 𝑞 sự kiện xảy ra, mỗi sự kiện thuộc một trong hai loại:

1. Các bạn học sinh *al* ,*al*1,...,*ar* cùng nhau tính lại điểm và nhận ra rằng điểm

ước lượng của mình “tăng” lên 𝑥 điểm.

1. Thầy giáo X muốn tính lại độ chênh lệch điểm của lớp.

***Yêu cầu:*** Với mỗi sự kiện loại 2, hãy tính giúp thầy độ chênh lệch của lớp.

**Dữ liệu vào:** Từ file văn bản DIVISION.INP:

* Dòng đầu chứa hai số nguyên *n*,*q*(1  *n*,*q*  2105) lần lượt là số học sinh và số sự kiện.
* Dòng hai chứa 𝑛 số nguyên *a*1, *a*2 ,..., *an* ( *a*  108) mô tả kì vọng các học sinh.

*i*

* *q* dòng sau, mỗi dòng thuộc một trong hai loại:
  + 1 *l r x* (1  *l*  *r*  *n*, *x*  108) tương ứng với sự kiện loại 1.
  + 2 tương ứng với sự kiện loại 2.

**Kết quả ra:** Xuất ra file văn bản DIVISION.OUT: Với mỗi sự kiện loại 2, in ra đáp án trên một dòng.

# Ràng buộc:

* Có 30% số test thỏa mãn 1  *n*, *q*  200
* Có 30% số test thỏa mãn 1  *n*, *q*  1000
* Có 40% số test không có ràng buộc gì thêm.

# Ví dụ:

|  |  |  |
| --- | --- | --- |
| **DIVISION.INP** | **DIVISION.OUT** | **Giải thích** |
| 4 7  -1 3 -6 8  1 4 4 2  2 | 20  23  23 | Với sự kiện loại 2 đầu tiên, điểm kì vọng của lớp là [−1,3,−6,10]  Điểm của các bạn làm bài tốt là [3,10], còn điểm của các bạn làm bài |

|  |  |  |
| --- | --- | --- |
| 1 4 4 3  2  1 1 2 -1  1 1 2 -4  2 |  | không tốt là [−1,−6], độ chênh lệch là 3+10−(−1−6)=20 |

* + 1. Phân tích bài toán

**Sub 1:** 𝟏 ≤ 𝒏,𝒒 ≤ 𝟐𝟎𝟎

Nhận thấy trong đáp án tối ưu, với mỗi nhóm nếu ta chọn học sinh có điểm cao nhất vào danh sách làm bài tốt và học sinh điểm thấp nhất vào danh sách làm bài không tốt.

Gọi 𝑑𝑝𝑖 là chênh lệch điểm nếu chỉ xét các học sinh từ 1 đến 𝑖. Ta có công thức:



Đáp án cho mỗi truy vấn 2 sẽ là 𝑑𝑝𝑛

Độ phức tạp là *O*(*q*  *n*2 )

**Sub 2:** 𝟏 ≤ 𝒏,𝒒 ≤ 𝟏𝟎𝟎𝟎

Nhận xét: Trong cách chia tối ưu, một nhóm học sinh được chọn 𝑙,𝑙 +1,…,𝑟 sẽ thỏa

mãn điều kiện *al*  *al*1  ...  *ar* hoặc *al*  *al*1  ...  *ar* .

Điều này là bởi nếu tồn tại một nhóm không thỏa mãn điều kiện trên, ta có thể nhóm đó chia thành các nhóm nhỏ hơn như trên mà tổng độ lệch không giảm đi.

Theo đó, độ lệch của một nhóm sẽ là *al* - *al*1  *al*1 - *al*2 ...  *ar*-1 - *ar* .

Từ đây ta sẽ xét hàm mục tiêu 𝑑𝑝[𝑖][𝑗 = 0/1] là độ lệch tối ưu trong cách chia các học sinh từ 1 → 𝑖, trong đó:

Nếu 𝑗 = 0, học sinh 𝑖 nằm trong một nhóm không giảm Nếu 𝑗 = 1, học sinh 𝑖 thuộc nhóm không tăng

Ta có một vài trường hợp:

* Học sinh thứ 𝑖 bắt đầu một nhóm mới: *dp*[*i*][ *j*]  max(*dp*[*i* 1][0], *dp*[*i* 1][1])
* Học sinh thứ 𝑖 tiếp tục nhóm trước:



Độ phức tạp là *O*(*q*  *n*)

**Sub 3:** Không có ràng buộc gì thêm.

Ta sẽ dựng segment tree, mỗi node quản lí đoạn [𝑙,𝑟] lưu mảng 𝑓[𝑖][𝑗] (0 ≤ 𝑖,𝑗 ≤ 1) với ý nghĩa

là độ chênh lệch trong đoạn [𝑙,𝑟]:

* Nếu 𝑖 = 0 thì 𝑙 thuộc nhóm không giảm, ngược lại 𝑙 thuộc dãy không tăng
* Nếu 𝑗 = 0 thì 𝑟 thuộc nhóm dãy không giảm, ngược lại 𝑟 thuộc nhóm không tăng.

Nếu gộp 2 node với nhau ta chỉ cần kiểm tra bạn cuối cùng của node bên trái và bạn đầu tiên của node bên phải có thể gộp thành cùng một nhóm nếu thỏa mãn các điều kiện sau:

* Cả 2 bạn đều phải thuộc cùng một loại nhóm (không giảm hoặc không tăng)
* Nếu loại nhóm là không giảm thì bạn bên trái ≤ bạn bên phải
* Nếu loại nhóm là không tăng thì bạn bên trái ≥ bạn bên phải

Đáp án mỗi truy vấn loại 2 sẽ là giá trị lớn nhất tại gốc của Segment tree. Độ phức tạp là 𝑂(𝑞𝑙𝑜𝑔2(𝑛))

* + 1. Chương trình minh họa

#define TASKNAME "DIVISION"

#include <bits/stdc++.h> using namespace std;

#define db(x) cerr << "[" << #x << " : " << x << "]" #define el cerr << "\n \n"

#define int long long

#define all(a) a.begin() + 1, a.end() #define alll(a) a.begin(), a.end()

#define fore(i, a, b) for(int i = (a); i <= (b); i++) #define sz(x) ((int) x.size())

#define fi first #define se second #define pb push\_back

using vi = vector<int>; using ii = pair<int, int>; const int maxn = 2e5 + 5; const int inf = 1e18;

int st[4 \* maxn][2][2], a[maxn]; void merge(int id, int mid)

{

fore(i, 0, 1)

fore(j, 0, 1)

{

st[id][i][j] = -inf; fore(u, 0, 1)

fore(v, 0, 1)

if (!u || !v || a[mid] \* a[mid + 1] >= 0) st[id][i][j] = max(

st[id][i][j], st[2 \* id][i][u] + st[2 \* id + 1][v][j]);

}

}

void update(int id, int l, int r, int pos)

{

if (l == r)

{

st[id][1][1] = abs(a[l]); return ;

}

int mid = (l + r) / 2;

if (pos <= mid) update(2 \* id, l, mid, pos); else update(2 \* id + 1, mid + 1, r, pos); merge(id, mid);

}

void solve()

{

int n, q;

cin >> n >> q;

fore(i, 1, n) cin >> a[i];

for (int i = n; --i; )

{

a[i + 1] -= a[i];

update(1, 1, n, i + 1);

}

a[1] = 0;

while (q--)

{

int op;

cin >> op; if (op == 2)

{

cout << max({st[1][0][0], st[1][0][1], st[1][1][0], st[1][1][1]}) << '\n';

continue;

}

int l, r, x;

cin >> l >> r >> x; if (l > 1)

{

a[l] += x; update(1, 1, n, l);

}

if (r < n)

{

a[r + 1] -= x; update(1, 1, n, r + 1);

}

}

}

signed main()

{

cin.tie(0)->sync\_with\_stdio(0);

if (fopen(TASKNAME ".inp", "r"))

{

freopen(TASKNAME ".inp", "r", stdin); freopen(TASKNAME ".out", "w", stdout);

}

solve();

return 0;

}

# Bài tập 13. PHÂN SỐ TĂNG (Mức độ khó)

* + 1. Đề bài

***Xét bài toán:*** Cho 𝑛 phân số 𝑎1 , … , 𝑎𝑛 (𝑎 , 𝑏

nguyên dương), hãy tìm dãy chỉ

𝑏1

𝑏𝑛

𝑖 𝑖

số 1 ≤ 𝑖 < 𝑖 < ⋯ < 𝑖 ≤ 𝑛 sao cho 𝑎𝑖1 < 𝑎𝑖2 < ⋯ < 𝑎𝑖𝑘 và 𝑘 lớn nhất có thể.

1 2 𝑘

𝑏𝑖1

𝑏𝑖2

𝑏𝑖𝑘

***Bài toán trên được mở rộng như sau:*** Có thể đảo lại một phân số nếu muốn (phân số 𝑎𝑖 đảo lại thành phân số 𝑏𝑖), sau đó tìm dãy chỉ số thỏa mãn đề bài mà 𝑘 lớn

𝑏𝑖

nhất có thể.

𝑎𝑖

***Yêu cầu:*** Cho 𝑛 phân số và số nguyên 𝑤, trong đó 𝑤 = 0 nghĩa là không được phép đảo bất kỳ một phân số nào (bài toán ban đầu) hoặc 𝑤 = 1 nếu được phép đảo không quá một phân số (bài toán mở rộng), hãy đưa ra giá trị 𝑘 lớn nhất có thể.

***Dữ liệu:*** Vào từ tệp văn bản FRACTION.INP theo khuôn dạng:

* Dòng đầu ghi hai số nguyên 𝑛, 𝑤(1 ≤ 𝑛 ≤ 103; 0 ≤ 𝑤 ≤ 1);
* Dòng thứ 𝑖 trong 𝑛 dòng tiếp theo chứa hai số nguyên dương 𝑎𝑖, 𝑏𝑖 lần lượt là tử số và mẫu số của phân số thứ 𝑖 (1 ≤ 𝑖 ≤ 𝑛, 0 < 𝑎𝑖, 𝑏𝑖 ≤ 109).

***Kết quả:*** Ghi ra tệp văn bản FRACTION.OUT một số nguyên duy nhất là giá trị 𝑘

lớn nhất tìm được.

## Ví dụ:

|  |  |
| --- | --- |
| **FRACTION.INP** | **FRACTION.OUT** |
| 4 0  5 1  1 3  3 2  1 2 | 2 |
| 4 1  5 1  1 3 | 3 |

|  |  |
| --- | --- |
| 3 2  1 2 |  |

***Ràng buộc:***

* + - * Có 25% số test ứng với 25% số điểm của bài thỏa mãn: 𝑛 ≤ 10, 𝑤 = 0;
      * Có 25% số test ứng với 25% số điểm của bài thỏa mãn: 𝑛 ≤ 10, 𝑤 = 1;
      * Có 25% số test ứng với 25% số điểm của bài thỏa mãn: 10 < 𝑛 ≤ 103, 𝑤 = 0;
      * Có 25% số test ứng với 25% số điểm của bài thoả mãn: 10 < 𝑛 ≤ 103, 𝑤 = 1.
    1. Phân tích bài toán
* **Subtask 1:** Duyệt 2n tập con, với mỗi tập con kiểm tra điều kiện tăng dần. Nếu thoả mãn thì cập nhật đáp án.
* **Subtask 2:** Duyệt vị trí thay đổi 𝑖 = 1 → 𝑛, với mỗi vị trí 𝑖 ta lại duyệt 2𝑛 tập con để kiểm tra điều kiện tăng dần. Lưu ý trường hợp không đổi con nào vẫn cho kết quả tốt nhất.

# Subtask 3:

* + Đây là bài toán quy hoạch động dãy con tăng dài nhất kinh điển.
  + Gọi 𝐹𝑖 là độ dài dãy con tăng dài nhất kết thúc tại 𝑖.
  + Ban đầu 𝐹𝑖 = 1 ∀𝑖 ∈ [1, 𝑛]
  + Công thức truy hồi 𝐹 = max (𝐹 + 1) nếu 𝑗 < 𝑖, 𝑎𝑗 < 𝑎𝑖, nghĩa là ta ghép 𝑖

𝑖 𝑗

𝑏𝑗

𝑏𝑖

vào sau dãy dài nhất kết thúc tại 𝑗 nếu có thể.

* + Kết quả là max (𝐹1, … , 𝐹𝑛)
  + Độ phức tạp: 𝑂(𝑛2). Ta có thể cải tiến xuống 𝑂(𝑛𝑙𝑜𝑔𝑛) bằng chặt nhị phân hoặc sử dụng các cấu trúc dữ liệu nâng cao

# Subtask 4

* + Ta quy hoạch động tương tự subtask 2 tuy nhiên cần lưu thêm chiều để kiểm soát điều kiện đảo ≤ 1 phân số
  + Gọi 𝐹(𝑖, 0 … 2) là số dãy con tăng kết thúc tại 𝑖 và
    - 𝑗 = 0: chưa đảo phân số nào
    - 𝑗 = 1: đã đảo một phân số, và phân số đó là 𝑎𝑖

𝑏𝑖

* + - 𝑗 = 2: đã đảo một phân số nhưng không phải là 𝑎𝑖

𝑏𝑖

* + Ban đầu 𝐹(𝑖, 𝑗) = 1 với mọi 𝑖 = 1 → 𝑛
  + Công thức truy hồi:
    - 𝐹(𝑖, 0) = max(𝐹(𝑗, 0) + 1) nếu 𝑗 < 𝑖, 𝑎𝑗 < 𝑎𝑖

𝑏𝑗 𝑏𝑖

* + - 𝐹(𝑖, 1) = max(𝐹(𝑗, 0) + 1) nếu 𝑗 < 𝑖, 𝑎𝑗 < 𝑏𝑖

𝑏𝑗 𝑎𝑖

* + - 𝐹(𝑖, 2) = max(𝐹(𝑗, 1) + 1) nếu 𝑗 < 𝑖, 𝑏𝑗 < 𝑎𝑖

𝑎𝑗 𝑏𝑖

* + - 𝐹(𝑖, 2) = max(𝐹(𝑗, 2) + 1) nếu 𝑗 < 𝑖, 𝑎𝑗 < 𝑎𝑖

𝑏𝑗 𝑏𝑖

* + Bởi vì nếu như 𝑖 là kết thúc của dãy tăng chưa đảo phân số nào thì cũng phải ghép vào dãy trước đó chưa đảo con nào. Còn nếu ta chọn đảo phân số 𝑖, thì trước đó cũng phải chưa đảo phân số nào. Cuối cùng, nếu đã đảo một phân số trước đó, thì phân số 𝑖 có thể là phân số nằm ngay sát phân số đã đảo, hoặc là không.
  + Kết quả là max(𝐹(𝑖, 𝑗)) với mọi bộ (𝑖, 𝑗) thoả mãn 𝑖 = 1 → 𝑛, 𝑗 = 0 → 2
  + Độ phức tạp: 𝑂(𝑛2)
    1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std; struct Fraction {

int x, y;

Fraction(int a = 0, int b = 0) { x = a;

y = b;

if (y == 0 || x == 0) { x = 0;

y = 1;

} else {

int gcd = gcd(x, y); x /= gcd;

y /= gcd; if (y < 0)

x = -x, y = -y;

}

}

bool operator<(const Fraction &rhs) const { return 1ll \* x \* rhs.y < 1ll \* y \* rhs.x;

}

};

Fraction a[1005];

Fraction b[1005]; int f[1005][3];

int main() { ios\_base::sync\_with\_stdio(0); cin.tie(0);

cout.tie(0);

#define task "FRACTION" if (fopen(task".inp", "r")) {

freopen(task".inp", "r", stdin);

freopen(task".out", "w", stdout);

}

int n, w;

cin >> n >> w; int res = 1;

for (int i = 1; i <= n; i++) { int x, y;

cin >> x >> y;

a[i] = Fraction(x, y);

b[i] = Fraction(a[i].y, a[i].x);

}

for (int i = 1; i <= n; i++) { f[i][0] = f[i][1] = f[i][2]=1;

for (int j = 1; j < i; j++) { if (a[j] < a[i]) {

f[i][0] = max(f[i][0], f[j][0] + 1);

f[i][2] = max(f[i][2], f[j][2] + 1);

}

if (a[j] < b[i]) {

f[i][1] = max(f[i][1], f[j][0] + 1);

}

if (b[j] < a[i]) {

f[i][2] = max(f[i][2], f[j][1] + 1);

}

}

if (!w) res = max(res, f[i][0]);

else res = max(res, \*max\_element(f[i], f[i] + 3));

}

cout << res;

}

# Cải tiến bài toán quy hoạch động bằng kỹ thuật chia để trị

Tiếp cận bài toán quy hoạch động mở rộng kỹ thuật chia để trị là cách tiếp cận với hai kỹ thuật chính là dùng đệ quy có nhớ kết hợp lập bảng quy hoạch động dựa trên công thức truy hồi. Khi đó chúng ta có thể vừa lưu trữ và thực hiện các giải pháp của các bài toán giúp nâng cao hiệu suất thực hiện bài toán (xử lý tối ưu).

Ví dụ xét bài toán Fibonaci:

- Sử dụng đệ quy với độ phức tạp O(2n) bằng cách tiếp cận top – down

Fib(n)

{

if (n < 2) result = n

else result = Fib(n-2) + Fib(n-1)

F[n] = result return F[n]

}

- Tiếp cận bottom up bằng phương pháp lập bảng quy hoạch động để giảm thời gian thực hiện thuật toán với độ phức tạp O(n) như sau:

tabFib(n)

{

F[0] = 0

F[1] = 1

for i = 2...n

F[i] = F[i-2] + F[i-1]

return F[n]

}

Ý tưởng chính của bài toán quy hoạch động được mở rộng kỹ thuật chia để trị là chúng ta phải xác định được bài toán lập bảng phương án quy hoạch động dựa trên công thức truy hồi nào và chia để trị ở đâu khi đó việc lưu trữ giá trị và sử dụng chúng để tính cho các thời điểm của chương trình khi thực hiện nó được tối ưu hơn.

# Bài tập 14. Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) (Mức độ khó)

* + 1. Đề bài

Khi xây dựng cầu thang đến các ngôi chùa nổi tiếng ở trên đỉnh núi, Chính quyền địa phương đã xác định N vị trí dọc theo sườn núi với các độ cao a1, a2, …, an. Trong đó

ai< ai+1 và 0 < *i* < *N*

Giá để xây dựng cầu thang từ vị trí đến vị trí j là:

𝑗

𝑚𝑖𝑛𝑣∈𝑍 ∑ |𝑎𝑠 − 𝑣|𝑘

𝑠=𝑖

Để đẩy nhanh quá trình xây dựng cầu thang từ vị trí 1 đến vị trí N, chính quyền địa phương đã quyết định giao việc cho G nhà xây dựng để xây dựng cầu thang song song nhau. Với N vị trí sẽ được chia thành G đoạn khác nhau và mỗi đoạn sẽ được phụ trách bởi một nhà thầu xây dựng khác nhau.

Với G nhà thầu xây dựng (0 < G≤ N) bạn hãy phân chia để G nhà thầu xây dựng cây cầu với tổng chi phí bé nhất.

Dữ liệu vào : PAGODA.INP có cấu trúc sau:

* Dòng 1 ghi ba số nguyên N, G, K (1£ *N* £ 2000,1£ *G* £ *N*,1£ *k* £ 2)

(1£ *a* £ 106, *a* £ *a* " 0< *i* < *N*)

* Dòng 2 ghi dãy số nguyên a1 , a2, … ,an vị trí cần xây dựng.

*i i i*+ 1

các

Kết quả ra: ghi vào file PAGODA.OUT giá trị xây dựng bé nhất.

# Ví dụ

|  |  |
| --- | --- |
| PAGODA.INP | PAGODA.OUT |
| 5 1 2  1 2 3 4 5 | 10 |
| 5 1 1  1 2 3 4 5 | 6 |
| 5 2 2  1 2 3 4 5 | 3 |

* + 1. Phân tích bài toán

Đặt cost(i, j) là chi phí để xây cầu thang từ điểm i đến j.

Đặt f(k, i) = chi phí nhỏ nhất để k nhóm thợ xây tất cả cầu thang từ 1 đến i. Ta có công thức QHĐ:

* + - * f(k, i) = min( f(k-1, j) + cost(j+1, i), với j = 1..i-1). Kết quả của bài toán chính là f(G, N).

Có 2 điểm mấu chốt của bài này:

* + - * Tính cost(i, j)
      * Tính nhanh bảng f(k, i). Nếu tính trực tiếp theo công thức trên, ta mất độ phức tạp O(G\*N2), không đủ để AC bài này.

1. Tính cost(i, j) Với k = 1:
   * Khi k = 1, điểm v chính là median của dãy A(i)..A(j).
   * Do dãy A đã được sắp xếp tăng dần, v = A[i + (j - i + 1) / 2]
   * Với mỗi (i, j), ta có thể tính nhanh cost(i, j) trong O(1) bằng [mảng cộng dồn](http://vnoi.info/wiki/algo/data-structures/data-structures-overview" \l "2-ctdl-truy-v%E1%BA%A5n_2-1-m%E1%BA%A3ng-c%E1%BB%99ng-d%E1%BB%93n-prefix-sum). Với k = 2:
   * Đặt L = số phần tử của đoạn A(i)..A(j) = j - i + 1
   * cost(i, j) = sum( (a(s) - v)2 )

o = sum( a(s)^2 ) + L\*v2 + 2\*sum( a(s) )\*v

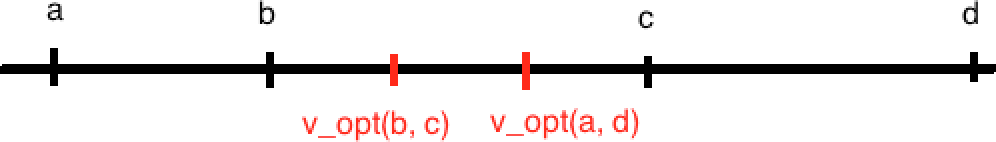
* + Đặt B = 2\*sum( a(s) ), C = sum( a(s)2 ). Ta tính được B và C trong O(1) bằng [mảng cộng dồn](http://vnoi.info/wiki/algo/data-structures/data-structures-overview" \l "2-ctdl-truy-v%E1%BA%A5n_2-1-m%E1%BA%A3ng-c%E1%BB%99ng-d%E1%BB%93n-prefix-sum).
  + cost(i, j) = L\*v2 + B\*v + C, là một hàm bậc 2 của v. Do đó điểm v để làm cost(i, j) nhỏ nhất chính là v = -B / L. Tuy nhiên (-B / L) có thể là số thực, còn trong bài toán này v phải là số nguyên, nên ta cần xét 2 số nguyên v gần nhất với (-B / L) bởi vì đồ thị hàm số lồi

1. Tính f(k, i)

Để tính f(k, i), ta cần dùng kĩ thuật [QHĐ chia để trị](http://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong" \l "2-chia-%C4%91%E1%BB%83-tr%E1%BB%8B). Ta có thể sử dụng được QHĐ chia để trị nếu:

* + cost(a, d) + cost(b, c) >= cost(a, c) + cost(b, d) với mọi a < b < c < d Đặt:
  + f(i, j, v) = (sum (|a(s) - v|k) với s = i..j)
  + v\_opt(a, b) = v để f(a, b, v) đạt giá trị nhỏ nhất
  + cost(a, b) = f(a, b, v\_opt(a, b)) <= f(a, b, v) với mọi v.

Không làm mất tính tổng quát, giả sử v\_opt(b, c) <= v\_opt(a, d). Ngoài ra ta cũng biết rằng v\_opt(b, c) nằm trong khoảng [b, c].



Ta có:

cost(a, d)

= f(a, d, v\_opt(a, d))

= f(a, b-1, v\_opt(a, d)) + f(b, d, v\_opt(a, d))

>= f(a, b-1, v\_opt(a, d)) + cost(b, d)

Mặt khác, do tất cả đoạn [a, b-1] ở bên trái v\_opt(b, c) và v\_opt(b, c) < v\_opt(a, d) nên: f(a, b-1, v\_opt(a, d)) + cost(b, c)

>= f(a, b-1, v\_opt(b, c)) + cost(b, c)

= f(a, b-1, v\_opt(b, c)) + f(b, c, v\_opt(b, c))

= f(a, c, v\_opt(b, c))

>= cost(a, c)

Kết hợp 2 bất đẳng thức trên:

cost(a, d) + cost(b, c)

>= f(a, b-1, v\_opt(a, d)) + cost(b, d) + cost(b, c)

= f(a, b-1, v\_opt(a, d)) + cost(b, c) + cost(b, d)

>= cost(a, c) + cost(b, d)

Chứng minh hoàn tất. Kết luận ta có thể dùng QHĐ chia để trị để giải bài toán với độ phức tạp O(G\*N\*log(G))

* + 1. Chương trình minh họa

#include <bits/stdc++.h> using namespace std; const int maxn = 2002; int n, g, k;

int a[maxn];

long long cost[maxn][maxn];

inline long long calc(long long a, long long b, long long c) { if (b % (2 \* a) == 0) {

long long x = b / (2 \* a); return a \* x \* x - b \* x + c;

} else {

long long x = b / (2 \* a); long long y = x + 1;

return min(a \* x \* x - b \* x + c, a \* y \* y - b \* y + c);

}

}

void prepare() { if (k == 1) {

for (int i = 1; i <= n; ++i) { long long cur\_cost = -a[i]; for (int j = i; j <= n; ++j) { cur\_cost += a[j];

cost[i][j] = cur\_cost; cur\_cost -= a[j + i + 1 >> 1];

}

}

} else {

for (int i = 1; i <= n; ++i) {

long long sa = 0, ssa = 0, s = 0; for (int j = i; j <= n; ++j) {

sa += a[j];

ssa += (long long)a[j] \* a[j];

s++;

cost[i][j] = calc(s, 2 \* sa, ssa);

}

}

}

}

long long f[maxn][maxn];

void divide(long long f[], long long g[], int l, int r, int pl, int pr) { if (l > r) {

return;

}

int m = (l + r) / 2; int ptr;

for (int i = pl; i < min(m, pr + 1); ++i) { long long val = f[i] + cost[i + 1][m];

if (val <= g[m]) { g[m] = val;

ptr = i;

}

}

divide(f, g, l, m - 1, pl, ptr);

divide(f, g, m + 1, r, ptr, pr);

}

int main() { #ifdef LOCAL

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout); #endif // LOCAL scanf("%d%d%d", &n, &g, &k); for (int i = 1; i <= n; ++i) {

scanf("%d", a + i);

}

prepare();

memset(f, 0x3f, sizeof(f)); f[0][0] = 0;

for (int i = 1; i <= g; ++i) { divide(f[i - 1], f[i], i, n, i - 1, n - 1);

}

printf("%lld\n", f[g][n]); return 0;

}

# Bài tập 15. SEQPART (Mức độ khó)

* + 1. Đề bài

Cho dãy L gồm các số C[1..L], cần chia dãy này thành G đoạn liên tiếp. Với phần tử thứ i, ta định nghĩa chi phí của nó là tích của Ci và số lượng các số nằm cùng đoạn liên tiếp với Ci. Chi phí của dãy số ứng với một cách phân hoạch là tổng các chi phí của các phần tử của G đoạn đã chia.

Yêu cầu: Hãy xác định cách phân hoạch dãy số để chi phí là nhỏ nhất.

**Dữ liệu vào** : tệp SEQPART.INP có cấu trúc sau

* Dòng đầu tiên chứa 2 số L và G.
* L dòng tiếp theo, chứa giá trị của dãy C1, C2, …, Cn.

**Kết quả ra**: ghi vào tệp SEQPART.OUT một dòng duy nhất là chi phí nhỏ nhất. Ràng buộc:

Ví dụ:

- 1 ≤ L ≤ 8000.

- 1 ≤ G ≤ 800.

- 1 ≤ Ci ≤ 109.

|  |  |
| --- | --- |
| **SEQPART.INP** | **SEQPART.OUT** |
| 6 3  11  11  11  24  26  100 | 299 |

Giải thích: Cách tối ưu là C[]=(11,11,11), (24,26), (100). Chi phí: 11∗3 + 11∗3 + 11∗3 + 24∗2 + 26∗2 + 100∗1 = 299.

* + 1. Phân tích bài toán

Đây là dạng bài toán phân hoạch dãy số có thể dễ dàng giải bài QHĐ. Gọi F(g,i) là chi phí nhỏ nhất nếu ta phân hoạch i phần tử đầu tiên thành g nhóm, khi đó kết quả bài toán sẽ là F(G,L).

Để tìm công thức truy hồi cho hàm F(g,i), ta sẽ quan tâm đến nhóm cuối cùng. Coi phần tử 0 là phần tử cầm canh ở trước phần tử thứ nhất, thì người cuối cùng không thuộc nhóm cuối có chỉ số trong đoạn [0,i][0,i]. Giả sử đó là người với chỉ số k, thì chi phí của cách phân hoạch sẽ là F(g−1,k)+Cost(k+1,i) với Cost(i,j) là chi phí nếu phân j−i+1 người có chỉ số [i,j]vào một nhóm. Như vậy:

F(g,i)=min(F(g−1,k)+Cost(k+1,l))với 0<=k<=i.

Chú ý là công thức này chỉ được áp dụng với g>1, nếu g=1,F(1,i)=Cost(1,i)đây là trường hợp cơ sở.

Chú ý là ta sử dụng mảng sum[] tiền xử lí O(L) để có thể truy vấn tổng một đoạn (dùng ở hàm cost()) trong O(1). Như vậy độ phức tạp của thuật toán này là O(G∗L∗L).

* Thuật toán tối ưu hơn

Gọi P(g,i) là k nhỏ nhất để cực tiểu hóa F(g,i), nói cách khác P(g,i])là k nhỏ nhất mà F(g,i)=F(g−1,k)+Cost(k+1,i).

Tính chất quan trọng để có thể tối ưu thuật toán trên là dựa vào tính đơn điệu của P(g,i), cụ thể: P(g,0) ≤ P(g,1) ≤ P(g,2) ≤ ⋯ ≤ P(g,L−1) ≤ P(g,L).

* Chia để trị

Để ý rằng để tính F(g,i), ta chỉ cần quan tâm tới hàng trước F(g−1) của ma trận: F(g−1,0),F(g−1,1),...,F(g−1,L).

Như vậy, ta có thể tính hàng F(g) theo thứ tự bất kỳ.

Ý tưởng là với hàng g, trước hết ta tính F(g,mid)F(g,mid) và P(g,mid) với mid=L/2, sau đó sử dụng tính chất nêu trên P(g,i)≤P(g,mid) với i<mid và P(g,i)≥P(g,mid) với i>mid để đi gọi đệ quy đi tính hai nửa còn lại.

* + 1. Chương trình minh họa

#include <bits/stdc++.h> const int MAXL = 8008; const int MAXG = 808;

const long long INF = (long long)1e18; using namespace std;

long long F[MAXG][MAXL], sum[MAXL], C[MAXL]; int P[MAXG][MAXL];

long long cost(int i, int j) { if (i > j) return 0;

return (sum[j] - sum[i - 1]) \* (j - i + 1);

}

void divide(int g, int L, int R, int optL, int optR) { if (L > R) return;

int mid = (L + R) / 2; F[g][mid] = INF;

for (int i = optL; i <= optR; ++i) {

long long new\_cost = F[g - 1][i] + cost(i + 1, mid); if (F[g][mid] > new\_cost) {

F[g][mid] = new\_cost; P[g][mid] = i;

}

}

divide(g, L, mid - 1, optL, P[g][mid]);

divide(g, mid + 1, R, P[g][mid], optR);

}

int main() { int G, L;

cin >> L >> G;

for (int i = 1; i <= L; ++i) { cin >> C[i];

sum[i] = sum[i - 1] + C[i];

}

for (int i = 1; i <= L; ++i) F[1][i] = cost(1, i); for (int g = 2; g <= G; ++g) divide(g, 1, L, 1, L); cout << F[G][L] << endl;

return 0;

}

# Một số bài khác

1. <https://oj.vnoi.info/problem/bedao_m22_e> Bedao Mini Contest 22 - Chia kẹo
2. <https://oj.vnoi.info/problem/lqdbus> Đến trường
3. <https://oj.vnoi.info/problem/nkcable> Nối mạng
4. <https://oj.vnoi.info/problem/nkrez> Hội trường
5. <https://oj.vnoi.info/problem/qtseq> Dãy số QT
6. <https://oj.vnoi.info/problem/quad> Xây hàng rào
7. <https://oj.vnoi.info/problem/qbmax> Đường đi có tổng lớn nhất
8. <https://oj.vnoi.info/problem/ctncln> Chia nhóm
9. <https://oj.vnoi.info/problem/disney1> Công viên Disneyland (version 1)
10. <https://oj.vnoi.info/problem/disney2> Công viên Disneyland (version 2)

…

# Kết luận

Bài toán tối ưu là một trong những lớp bài toán có nhiều ứng dụng trong nghiên cứu và thực tế. Kĩ thuật quy hoạch động là dạng hay gặp trong các bài toán tối ưu. Chính vì vậy việc rèn tư duy cho học sinh khối chuyên có thể áp dụng được trong khi lập trình cho các bài toán đặc trưng là rất quan trọng vầ cần thiết. Chuyên đề trên là một số vấn đề cần dạy các tư duy và tiếp cận cho học sinh chuyên khi bắt đầu học về quy hoạch động. Mở rộng bài toán quy hoạch động với kĩ thuật chia để trị là cách để giảm độ phức tạp khi lập trình với các bài toán quy hoạch động.

Do kinh nghiệm chưa nhiều, nên chắc chắn có thiếu sót. Tôi rất mong đồng nghiệp và bạn bè chia sẻ, góp ý thêm cho tôi. Tôi xin chân thành cảm ơn!

# Nguồn tài liệu tham khảo

[http://vnoi.info](http://vnoi.info/) [https://vn.spoj.com](https://vn.spoj.com/) [https://www.geeksforgeeks.org](https://www.geeksforgeeks.org/) [https://codeforces.com](https://codeforces.com/) [https://www.codechef.com](https://www.codechef.com/)