HỘI THẢO KHOA HỌC CÁC TRƯỜNG THPT CHUYÊN DUYÊN HẢI ĐỒNG BẰNG BẮC BỘ NĂM 2019

----------*----------**

CHUYÊN ĐỀ

KỸ THUẬT BAO LỒI

MỤC LỤC

A. MỞ ĐẦU

Quy hoạch động là một trong những thuật toán hiệu quả và được lựa chọn sử dụng nhiều trong các kỳ thi lập trình. Bởi vì tính phổ dụng của nó nên ngày càng có nhiều nghiên cứu để đưa quy hoạch động trở nên gần gũi, quen thuộc với người lập trình đồng thời cải tiến để các thuật toán sử dụng quy hoạch động ngày một tối ưu hơn. Kỹ thuật bao lồi (convex hull trick) là một trong những cải tiến đó.

Kĩ thuật bao lồi là kĩ thuật (hoặc là cấu trúc dữ liệu) dùng để xác định hiệu quả, có tiền xử lý, cực trị của một tập các hàm tuyến tính tại một giá trị của biến độc lập. Mặc dù tên gọi giống nhưng kĩ thuật này lại khá khác biệt so với thuật toán bao lồi của hình học tính toán. Kỹ thuật này được dùng để cải tiến thuật toán quy hoạch động với thời gian $O(n^2)$ xuống còn O(nlogn), thậm chí với một số trường hợp xuống còn O(n) cho một lớp các bài toán.

Đây là một kỹ thuật khá lạ và ít có nguồn trên mạng về vấn đề này, nên chuyên đề của tôi được trình bày với mục đích có thể tổng hợp lại một số kiến thức cơ bản đã có của kỹ thuật bao lồi, qua đó là một tài liệu tham khảo cho các bạn quan tâm đến nội dung trên.

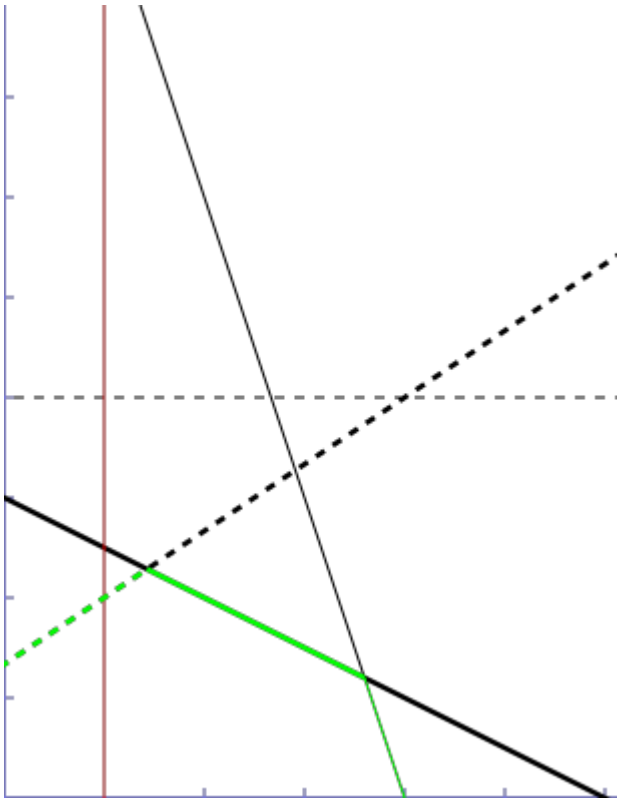
B. NỘI DUNG

1. Bài toán cơ bản:

Cho một tập lớn các hàm tuyến tính có dạng $y=m_i x+b_i y$ *và một số lượng lớn truy vấn. Mỗi truy vấn là một số x và hỏi giá trị cực tiểu y có thể đạt được nếu chúng ta thế x vào một trong những phương trình đã cho.*

Ví dụ: cho các phương trình: y=4; y= 4/3+ 2/3 x; y= 12-3; y= 3-1/2x và truy vấn x=1. Khi đó giá trị cực tiểu y có thể đạt được là 2 và phương trình cho giá trị cực tiểu đó là y=4/3 +2/3x

Thật vậy, sau khi ta vẽ các đường thẳng lên hệ trục tọa độ, dễ thấy rằng: chúng ta muốn xác định, tại x=1(đường màu đỏ) đường nào có tọa độ y nhỏ nhất. Ở trong trường hợp này là đường nét đứt đậm y=4/3+2/3x.



2. Ý tưởng thuật toán:

a.Thuật toán thông thường:

Với mỗi truy vấn trong Q truy vấn, ta duyệt qua từng hàm số và thử từng hàm xem hàm nào trả giá trị cực tiểu cho giá trị x. Nếu có M đường thẳng và Q truy vấn, độ phức tạp của thuật toán sẽ O(MQ).

Kĩ thuật bao lồi sẽ giúp giảm độ phức tạp xuống còn O((Q+M)logM), một độ phức tạp hiệu quả hơn nhiều.

b. Kỹ thuật bao lồi:

Xét hình vẽ ở trên. Đường thẳng y=4 sẽ không bao giờ là giá trị nhỏ nhất với tất cả giá trị của x. Mỗi đường trong mỗi đường thẳng còn lại sẽ lại trả lại giá trị cực tiểu trong một và chỉ một đoạn liên tiếp (có thể có một biên là $+\infty$ hoặc $-\infty$). Đường chấm đậm sẽ cho giá trị cực tiểu với tất cả giá trị nằm bên trái điểm giao với đường đen đậm. Đường đen đậm sẽ cho giá trị cực tiểu với tất cả giá trị giữa giao điểm của nó với đường nhạt và đường chấm đậm. Và đường nhạt sẽ nhận cực tiểu cho tất cả giá trị bên phải giao điểm với đường đậm. Một nhận xét nữa là với giá trị của x càng tăng thì hệ số góc của các hàm số sẽ giảm: $2/3, -1/2, -3$. Với một chút chứng minh dễ thấy rằng điều này luôn đúng.Điều này giúp chúng ta hiểu phần nào thuật toán:

* Bỏ đi các đường thẳng không quan trọng như y=4 trong ví dụ (những đường thẳng mà không nhận giá trị cực tiểu trong bất kì đoạn nào)

* Sắp xếp các đoạn thẳng còn lại theo hệ số góc và được một tập N đoạn thẳng (của N đường thẳng còn lại) mà mỗi đoạn có một đường thẳng nhận giá trị cực tiểu.

* Dùng thuật toán tìm kiếm nhị phân cơ bản để có thể tìm kiếm đáp án cho từng truy vấn.

Trong ví dụ, nếu chúng ta coi mỗi phần đoạn thẳng tối ưu của đường thẳng (bỏ qua đường y=4), chúng ta sẽ thấy những đoạn đó tạo thành một *hình bao dưới* (lower envelope), một tập các đoạn thẳng chứa tất cả điểm cực tiểu cho mọi giá trị của x (hình bao dưới được tô bằng màu xanh trong hình). Cái tên *kĩ thuật bao lồi* xuất phát từ việc đường bao trên tạo thành một đường lồi, từ đó thành bao lồi của một tập điểm.

Ta có thể thấy nếu ta có một tập đường thẳng đã được xác định sắp xếp, ta có thể dễ dàng trả lời bất kì truy vấn nào với độ phức tạp là O(logM) với tìm kiếm nhị phân. Vậy nếu chúng ta tìm ra cách thêm một đường thẳng vào tính toán lại một cách hiệu quả là chúng ta đã có một thuật toán hoạt động ngon lành.

Giả sử chúng ta được xử lý tất cả đường thẳng trước khi làm các truy vấn thì chúng ta chỉ cần đơn giản sắp xếp các đường thẳng theo hệ số góc và thêm từng đường một vào. Sẽ có thể một số đường không quan trọng và sẽ bị bỏ đi. Chúng ta sẽ sử dụng cấu trúc dữ liệu Stack để cài đặt, bỏ từng đường thẳng vào stack và nếu đường nào không quan trọng sẽ bị bỏ ra ngoài đến khi chỉ còn một đường thẳng

(đường thẳng cuối không thể bỏ).

Vậy làm sao để có thể xác định đường thẳng nào sẽ bị bỏ khỏi stack? Giả sử l1, l2 và l3 là đường thẳng áp chót (gần cuối) trên stack, đường thẳng cuối cùng trong stack và đường thẳng được thêm vào stack. Đoạn l2 không quan trọng (không có giá trị cực tiểu ở điểm nào) khi và chỉ khi giao điểm của l1 và l3 nằm bên trái giao điểm của l1 và l2 (đoạn mà l3 nhận giá trị cực tiểu đã nằm đè lên đoạn của l2). Giả sử rằng không có ba đường nào trùng hay song song với nhau (có thể giải quyết một cách đơn giản).

c. Độ phức tạp của thuật toán:

Độ phức tạp bộ nhớ sẽ là O(M): chúng ta cần một danh sách lưu lại các đoạn thẳng, biểu diễn bởi hai số thực.

Độ phức tạp thời gian cho phần tiền xây dựng:

- Để sắp xếp các đoạn thẳng theo tăng dần hệ số góc sẽ tốn O(MlogM)
- Duyệt qua các đường thẳng mỗi đường được cho vào stack và bỏ khỏi stack tối đa một lần vậy nên sẽ tốn O(M) cho bước này.

Vậy thời gian cho việc xây dựng sẽ là O(MlogM). Với mỗi truy vấn dùng chặt nhị phân sẽ cho độ phúc tạp tốt nhất O(logM).

Biến thể khác của bài toán có thể là tìm giá trị cực đại chúng ta làm hoàn toàn tương tự.

3. Bài tập vận dụng:

Bài tập 1: USACO Tháng 3 năm 2008 "Acquire"

Cho $N(N \leq 50000)$ hình chữ nhật khác nhau về hình dạng, mục tiêu của bài toán là phải lấy được tất cả hình chữ nhật. Một tập hình chữ nhật có thể thu được với chi phí bằng tích của chiều dài dài nhất và chiều rộng dài nhất. Chúng ta cần phân hoạch tập các hình chữ nhật này một cách khôn khéo sao cho tổng chi phí có thể được tối thiểu hóa và tính chi phí này. Hình chữ nhật không thể được xoay (đổi chiều dài và chiều rộng).

Solution:

*Ta có các nhận xét sau:

- Tồn tại các hình chữ nhật không quan trọng: Giả sử tồn tại hai hình chữ nhật A và

B mà mà cả chiều dài và chiều rộng của hình B đều bé hơn hình A thì ta có thể nói hình B là không quan trọng vì ta có thể để hình B chung với hình A từ đó chi phí của hình B không còn quan trọng. Sau khi đã loại hết tất cả hình không quan trọng đi và sắp xếp lại các hình theo chiều dài giảm dần thì chiều rộng các hình đã sắp xếp sẽ theo chiều tăng

- Đoạn liên tiếp:Sau khi sắp xếp, ta có thể hình dung được rằng nếu chúng ta chọn hai hình chữ nhật ở vị trí i và ở vị trí j thì ta có thể chọn tất cả hình chữ nhật từ i+1đến j−1 mà không tốn chi phí nào cả. Vậy ta có thể thấy rằng cách phân hoạch tối ưu là một cách phân dãy thành các đoạn liên tiếp và chi phí của một đoạn là bằng tích của chiều dài của hình chữ nhật đầu tiên và chiều rộng của hình chữ nhật cuối cùng.

* Giải bài toán theo phương pháp quy hoạch động bình thường: bài toán trở về bài toán phân dãy sao cho tổng chi phí của các dãy là tối ưu. Đây là một dạng bài quy hoạch động hay gặp và chúng ta có thể dễ dàng nghĩ ra thuật toán $O(N^2)$ như mã giả phía dưới. (Giả sử các hình đã được sắp xếp và bỏ đi những hình chữ nhật không quan trọng)

```
input N
for i ∈ [1..N]
     input rect[i].h
     input rect[i].w
let cost[0] = 0
for i ∈ [1..N]
     let cost[i] = ∞
     for j ∈ [0..i-1]
         cost[i] = min(cost[i],cost[j]+rect[i].h*rect[j+1].w)
print cost[N]
```

Ở trên cost[k] lưu lại chi phí cực tiểu để lấy được k hình chữ nhật đầu tiên. Hiển nhiên, cost[0]=0. Để tính toán được cost[i] với i khác 0, ta có tính tổng chi phí để lấy được các tập trước và cộng nó với chi phí của tập cuối cùng(có chứa i). Chi phí của một tập có thể dễ dàng tính bằng cách lấy tích của chiều dài hình chữ nhật đầu tiên và chiều rộng của hình chữ nhật cuối cùng.

Vậy ta có min(cost[i],cost[j]+rect[i].h*rect[j+1].w) với j là hình chữ nhật đầu tiên của tập cuối cùng. Với N=50000 thì thuật toán $O(N^2)$ này là quá chậm.

*Để cải tiến thuật toán trên, ta sử dụng kỹ thuật bao lồi như sau:

Với m_j=rect[j+1].w,b_j=cost[j],x=rect[i].h, với rect[x].h là chiều rộng của hình chữ nhật x và rect[x].w là chiều dài của hình chữ nhật x. Vậy thì bài toán trở về tìm hàm cực tiểu của $y=m_jx+b_j$ bằng cách tìm j tối ưu. Nó giống hoàn toàn bài toán chúng ta đã đề cập ở trên. Giả sử ta đã hoàn thành việc cài đặt cấu trúc đã đề cập ở trên chúng ta có thể có mã giả ở dưới đây:

```
input N
for i ∈ [1..N]
    input rect[i].h
    input rect[i].w
let E = empty lower envelope structure
let cost[0] = 0
add the line y=mx+b to E, where m=rect[1].w and b=cost[0] //b is zero
for i ∈ [1..N]
     cost[i] = E.query(rect[i].h)
    if i<N
        E.add(m=rect[i+1].w,b=cost[i])
print cost[N]
```

Rõ ràng các đường thẳng đã được sắp xếp giảm dần về độ lớn của hệ số góc do chúng ta đã sắp xếp các chiều dài giảm dần. Do mỗi truy vấn có thể thực hiện trong thời gian O(logN), ta có thể dễ dàng thấy thời gian thực hiện của cả bài toán là O(NlogN). Do các truy vấn của chúng ta cũng tăng dần (do chiều rộng đã được sắp xếp tăng tần) ta có thể thay thế việc chặt nhị phân bằng một con trỏ chạy song song với việc quy hoạch động đưa bước quy hoạch động còn O(N) nhưng tổng độ phức tạp vẫn là O(NlogN) do chi phí sắp xếp.

Code tham khảo: http://wcipeg.com/wiki/Convex_hull_trick/acquire.cpp

Bài tập 2: APIO 2010 Commando

Cho một dãy có N số nguyên (1≤N≤106) và một hàm số bậc 2 với hệ số nguyên dương $f(x)=ax^2+bx+c,a<0$. Hãy phân dãy này ra thành các đoạn liên tiếp sao tổng các hàm f trên các dãy là lớn nhất (giá trị của hàm f lên dãy là f(x) với x là tổng dãy đó) .

Solution:

-Tương tự như bài trên công thức quy hoạch động có thể dễ thấy công thức O(N2)

Định nghĩa rằng:

`sum(i,j) = x[i] + x[i+1] + ... + x[j]`

`adjust(i,j) = a*sum`^{2}`(i,j) + b*sum(i,j) + c`

Ta có:

$dp(n)=max^{n-1}_{k=0}[dp(k)+adjust(k+1,n)]$

Mã giả:

```
dp[0] ← 0
for n ∈ [1..N]
  for k ∈ [0..n-1]
    dp[n] ← max(dp[n], dp[k] + adjust(k + 1 , n))
```

Hãy thử biến đổi hàm "adjust" một tí nào. Định nghĩa sum(1,x) là $\delta(x)$. Vậy với một số k bất kì ta có thể viết là:

$$dp(n)=dp(k)+a(\delta(n)-\delta(k))^2+b(\delta(n)-\delta(k))+c$$

hay $dp(n)=dp(k)+a(\delta^2(n)+\delta^2(k)-2\delta(n)\delta(k))+b(\delta(n)-\delta(k))+c$

hay $dp(n)=(a\delta^2(n)+b\delta(n)+c)+dp(k)-2a\delta(n)\delta(k)+a\delta^2(k)-b\delta(k)$

Nếu:

- $z=\delta(n)$
- $m=-2a\delta(k)$
- $p=dp(k)+a\delta^2(k)-b\delta(k)$

Ta có thể thấy mz+p là đại lượng mà chúng ta muốn tối ưu hóa bằng cách chọn k , dp(n) sẽ bằng đại lượng đó cộng thêm với $a\delta(n)+b\delta(n)+c$ độc lập so với k).

Trong đó z cũng độc lập với k, và m, p phụ thuộc vào k.

Ngược với bài "acquire" khi chúng ta phải tối thiểu hóa hàm quy hoạch động thì bài này chúng ta phải cực đại hóa nó. Chúng ta phải xây dựng một hình bao trên với các đường thẳng tăng dần về hệ số góc. Do đề bài đã cho a<0 hệ số góc của chúng ta tăng dần và luôn dương thỏa với điều kiện của cấu trúc chúng ta.

Do dễ thấy $\delta(n)>\delta(n-1)$, giống như bài "acquire" các truy vấn chúng ta cũng tăng dần theo thứ tự do vậy chúng ta có thể khơi tạo một biến chạy để chạy song song khi làm quy hoạch động (bỏ được phần chặt nhị phân).

Code tham khảo: http://wcipeg.com/wiki/Convex_hull_trick/commando.cpp

Bài tập 3: http://codeforces.com/contest/319/problem/C

Cho 2 dãy số a và b nguyên đều có N phần tử a[1], a[2], ..., a[N] và b[1], b[2], .., b[N]. Mỗi lượt, bạn được chọn 1 số a[i] bất kỳ và giảm giá trị của a[i] đi 1, nhưng bạn sẽ mất 1 lượng năng lượng bằng b[j] với j là vị trí lớn nhất mà a[j] = 0(Bắt buộc phải có ít nhất 1 số a[j] = 0). Bài toán yêu cầu in ra số năng lượng nhỏ nhất để giảm tất cả a[i] về 0. Với mọi (i < j): (a[i] < a[j]); (b[i] > b[j]); (b[n] = 0) và (a[1] = 1). Ban đầu bạn sẽ có 1 lượt không tốn năng lượng.

***Input**: Dòng đầu tiên nhập N (1 <= N <= 10^5).*

Dòng thứ 2 nhập dãy a. (1 <= a[i] <= 10^9).

Dòng thứ 3 nhập dãy b. (1 <= b[i] <= 10^9).

***Output:** In ra số năng lượng nhỏ nhất để giảm dãy a về 0.*

***Ví dụ**: Input: 6*

1 2 3 10 20 30

6 5 4 3 2 0

Output: 138

(Giải thích:

Đầu tiên giảm số a[1] về 0, số năng lượng mất đi: 0 (lượt ban đầu)

*Giảm a[3] về 0, số năng lượng mất đi: 3 * 6 (do 1 là vị trí lớn nhất mà a[1] = 0, nên năng lượng mất là b[1] = 6).*

*Giảm a[6] về 0, số năng lượng mất đi: 30 * 4 (do 3 là vị trí lớn nhất mà a[3] = 0, nên năng lượng mất đi là b[3] = 4).*

Sau đó, có thể giảm bất kỳ số a[i] nào về 0 mà không tốn năng lượng (do 6 là vị trí lớn nhất mà a[6] = 0 và b[6] = 0).

*Vậy số năng lượng mất đi là : 3 * 6 + 30 * 4 = 138.)*

Solution:

Nhận xét : Ta có 1 lượt không tốn năng lượng, a[1] = 1 và bắt buộc phải có 1 số a[i] = 0 nên lượt đầu, ta chắc chắn phải giảm a[1] đi 1. Mục tiêu của ta là tốn ít năng lượng nhất để giảm a[n] về 0, vì b[n] = 0 và n là vị trí lớn nhất nên các số a[i] còn lại ta có thể giảm về 0 mà không tốn năng lượng.

Từ đây, ta có thể xây dựng mảng qhđ : f[i] là năng lượng nhỏ nhất để giảm a[i] về 0. Kết quả bài toán sẽ là f[n].

Công thức qhđ: f[i] = min(b[j] * a[i] + f[j]) (j chạy từ 1 đến i - 1).

Với cách làm trên, độ phức tạp sẽ là O(N^2).

Với N <= 10^5, cách này sẽ không đạt yêu cầu. Vì vậy ta phải cải thiến thuật toán như sau theo kỹ thuật bao lồi:

Ta có thể để ý công thức qhđ có dạng 1 hàm y = m * x + b với (y = f[i]; b = f[j]; m = b[j]; x = a[i]).

Và hệ số góc m = b[j] giảm dần, thỏa mãn điều kiện để kỹ thuật bao lồi để giảm đpt xuống O(N * log(N)).

Code tham khảo: http://codeforces.com/contest/319/submission/42019806

Bài tập 4: http://codeforces.com/contest/311/problem/B

Có M con mèo và có P người cho mèo ăn. Có N ngọn đồi trải dài từ trái qua phải (1, 2, 3, ... ,N). Khoảng cách giữa ngọn đồi (i) và ngọn đồi (i - 1) là d[i] mét. P người cho mèo ăn ban đầu đều ở ngọn đồi 1.Vào một ngày, các con mèo ra ngoài đi chơi. Con mèo i đến ngọn đồi h[i] vào thời điểm t[i] và ngồi chờ người cho mèo ăn đến.Mỗi người cho mèo ăn sẽ xuất phát tại một thời điểm bất kỳ, đi thẳng từ ngọn đồi 1 đến ngọn đồi N và sẽ bắt những con mèo ĐANG ĐỨNG CHỜ ở các ngọn đồi. Người cho mèo ăn đi 1 mét / 1 phút và có thể bắt bao nhiêu con mèo họ muốn.

Ví dụ, có 2 ngọn đồi, khoảng cách giữa 2 ngọn đồi là 1 mét (d[2] = 1) và có 1 con mèo kết thúc chuyến đi chơi của nó tại ngọn đồi 2 ở thời điểm phút thứ 3. Nếu người cho mèo ăn rời khỏi ngọn đồi 1 ở thời điểm 2 hoặc 3, anh ta có thể bắt được con mèo, nhưng nếu anh ta rời ngọn đồi 1 ở thời điểm 1 thì sẽ không bắt được con mèo. Và nếu anh ta rời ngọn đồi 1 ở thời điểm 2, con mèo sẽ phải chờ trong 0 phút. Nếu anh ta rời ngọn đồi 1 ở thời điểm 3, con mèo sẽ phải chờ 1 phút.

Yêu cầu: Sắp xếp lịch trình cho P người cho mèo ăn để tổng thời gian chờ của M con mèo là nhỏ nhất. In ra thời gian chờ đó.

***Input** : Dòng đầu nhập N, M, P. (2 <= N <= 10^5, 1 <= M <= 10^5, 1 <= P <= 100).*

Dòng tiếp theo nhập mảng d gồm N - 1 số là khoảng cách giữa 2 ngọn đồi liên tiếp. (1 <= d[i] < 10^4).

M dòng tiếp theo, dòng thứ i gồm h[i] và d[i]. (1 <= h[i] <= n, 0 <= t[i] <= 10^9).

***Output:** Tổng thời gian chờ nhỏ nhất của M con mèo*

Ví dụ:

Input: 4 6 2 *Output: 3*

1 3 5

1 0

2 1

4 9

1 10

2 10

3 12

(Giải thích:

Người thứ nhất sẽ rời ngọn đồi 1 ở thời điểm 0, bắt con mèo thứ nhất, đến ngọn đồi 2 ở thời điểm 1, bắt con mèo thứ 2, đến ngọn đồi 3 ở thời điểm 4, đến ngọn đồi 4 ở thời điểm 9, bắt con mèo thứ 3. Cả 3 con mèo này đều không tốn thời gian chờ.

Người thứ hai sẽ rời ngọn đồi 1 ở thời điểm 10, bắt con mèo thứ 4, đến ngọn đồi 2 ở thời điểm 11 bắt con mèo thứ 5(con mèo thứ 5 đến ngọn đồi 2 ở thời điểm 10 nên phải chờ 1 phút), đến ngọn đồi 3 ở thời điểm 14 bắt con mèo thứ 6(con mèo thứ 6 đến ngọn đồi 3 ở thời điểm 12 nên phải chờ 2 phút).

Vậy tổng số thời gian M con mèo phải chờ là 3 phút.

Solution: Đây là một bài rất khó. Chúng ta cần phải biến đổi một chút để làm bài toán dễ dàng hơn.

Gọi s[i] là tổng khoảng cách từ ngọn đồi 1 đến ngọn đồi i.

Công thức: s[i] = s[i - 1] + d[i].

Gọi a[i] là thời điểm đi sớm nhất của người cho ăn để bắt được con mèo thứ i.

Công thức a[i] = t[i] - s[h[i]].

Ta sort lại dãy a theo thứ tự tăng dần.

Bây giờ, nếu ta đi vào thời điểm a[i], chắc chắn ta sẽ bắt được các con mèo từ a[i] đến a[M].

=> Bài toán trở thành: Phân dãy a[i] thành P tập liên tiếp sao cho tổng thời gian chờ là nhỏ nhất.

Công thức: Giả sử chúng ta phân các con mèo [i, j] cho 1 người đi bắt. Vậy tức là anh ta phải đi vào thời điểm a[i] và tổng thời gian các chờ của các con mèo sẽ là:

(a[j + 1] - a[j]) + (a[j + 2] - a[j]) + … + (a[i] - a[j])

= (sum[i] - sum[j]) - (i - j) * a[j]. (Với sum là mảng tổng tiền tố của mảng a).

Từ đây, ta có thể dễ dàng xây dựng công thức qhđ:

Gọi F[k][i] : tổng thời gian chờ nhỏ nhất để bắt i con mèo với số người là k.

Kết quả bài toán sẽ là F[p][m].

Công thức: F[k][i] = F[k - 1][j - 1] + (sum[i] - sum[j]) - (i - j) * a[j].

F[k][i] = F[k - 1][j - 1] + sum[i] - sum[j]- i * a[j] + j * a[j].

F[k][i] = i * (-a[j]) + (F[k - 1][j - 1] - sum[j]+ j * a[j]) + sum[i].

Hay F[k][i] = x * m + b + c

Trong đó: i là x;-a[j] là m;F[k - 1][j - 1] - sum[j] + j * a[j] là b.

c là phần thêm vào, hoàn toàn xử lý được.

Và chúng ta có thể thấy hệ số góc (m = -a[j]) giảm dần (do dãy a[i] tăng dần nên giá trị -a[i] sẽ giảm dần).

Vậy là đủ điều kiện để áp dụng kỹ thuật bao lồi để giảm độ phức tạp và khi đó độ phức tạp của bài toán là O(M * P * log(M)).

Bài tập 5: http://codeforces.com/contest/660/problem/F

*Cho 1 dãy a gồm N số a[1], a[2], … a[N]. Nếu bạn chọn 1 dãy con liên tiếp trong dãy a, giả sử đoạn [i, j], số điểm bạn đạt được sẽ là: a[i] * 1 + a[i + 1] * 2 + … + a[j] * (j - i + 1).*

***Yêu cầu**: bạn cần chọn 1 dãy con liên tiếp trong dãy a sao cho số điểm bạn đạt được là lớn nhất. (Lưu ý: Bạn sẽ được 0 điểm nếu chọn dãy rỗng).*

***Input:** Dòng đầu tiên nhập N (1 <= N <= 2 * 10^5) - số lượng số trong dãy a.*

Dòng thứ 2 nhập N số a[1], a[2], … , a[N] (|a[i]| <= 10^7).

***Output:** In ra số điểm lớn nhất đạt được.*

Ví dụ:

Input: 6 *Output: 16*

5 -1000 1 -3 7 -8

Giải thích: Chúng ta sẽ chọn dãy liên tiếp: 1 -3 7.

*Số điểm nhận được sẽ là: 1 * 1 + (-3) * 2 + 7 * 3 = 16.*

Solution:

Gọi s[i] = a[1] + a[2] + a[3] + … + a[i].

Gọi d[i] = a[1] * 1 + a[2] * 2 + … + a[i] * i.

Với 2 mảng này, ta hoàn toàn có thể tính số điểm đạt được nếu chọn đoạn [j, i] trong O(1).

Công thức = (d[i] - d[j - 1]) - (s[i] - s[j - 1]) * (j - 1).

Với công thức này, ta hoàn toàn có thể xử lý bài toán này với độ phức tạp O(N^2).Gọi f[i] : số điểm lớn nhất đạt được khi ta chọn 1 dãy liên tiếp kết thúc ở vị trí i.

f[i] = max((d[i] - d[j - 1]) - (s[i] - s[j - 1]) * (j - 1)) (j chạy từ 1 -> i).

Kết quả sẽ là max(f[i]) (i chạy từ 1 -> N).

Với N <= 10^5. Ta phải cải tiến thuật toán. Ta sẽ biến đổi công thức f[i].

f[i] = (d[i] - d[j - 1]) - (s[i] - s[j - 1]) * (j - 1).

= d[i] - d[j - 1] - s[i] * (j - 1) + s[j - 1] *(j - 1).

= (-s[i]) * (j - 1) + (-d[j - 1] + s[j - 1] * (j - 1)) + d[i].

y = x * m + b + c

Ta có thể thấy, công thức của chúng ta lại có dạng giống như một hàm số, với hệ số góc m = (j - 1); b = -d[j - 1] + s[j - 1] * (j - 1); y = f[i]; x = -s[i].

c là hằng số đã biết, mình có thể cộng trực tiếp vào f[i].

Và hệ số góc m = (j - 1) tăng dần, thỏa mãn đầy đủ điều kiện để áp dụng kỹ thuật bao lồi.Khi đó độ phức tạp của thuật toán này là :O(N * log(N)).

Code tham khảo: http://codeforces.com/contest/660/submission/42116118

C. KẾT LUẬN

Kỹ thuật bao lồi là một trong những kỹ thuật khó và chưa có nhiều tài liệu đề cập đến. Để sử dụng kỹ thuật này yêu cầu hàm đang tìm phải có dạng y = m * x + b. (Có thể một số bài toán phải biến đổi nhiều bước mới ra được dạng này). Đồng thời hệ số góc m phải giảm dần nếu tìm hàm min (phải tăng dần nếu tìm hàm max). Đồng thời kỹ thuật chỉ áp dụng được với đường thẳng có dạng y = m * x + b.Nếu nó 1 đoạn thẳng, tức là đường thẳng y = m * x + b bị giới hạn x trong đoạn [a, b] thì phải dùng thuật toán IT đường thẳng. Ngoài ra, kỹ thuât này có thể dễ dàng được thực hiện khi các đường thẳng được thêm trước tất cả các truy vấn hay các đường thẳng được thêm vào theo thứ tự giảm dần của hệ số góc. Hoặc với cấu trúc deque chúng ta cũng có thể thêm những đường thẳng có hệ số góc lớn hơn hết các đường thẳng đã có. Nhưng có những lúc sẽ có các bài toán khi chúng ta phải giải quyết các truy vấn và thêm đường thẳng lồng vào nhau với các hệ số góc ngẫu nhiên. Chúng ta không thể sắp xếp lại trước (do bị lồng vào truy vấn) và không thể sắp xếp lại với mỗi lần thêm đường thẳng (vậy sẽ cho ta một độ phức tạp tuyến tính với mỗi truy vấn).Có tồn tại một cách để thêm các đường thẳng ngẫu nhiên vào trong độ phức tạp log. Chúng ta lưu các đoạn thẳng trong một cấu trúc có thứ tự động như std::set của C++. Mỗi đường thẳng chứa hệ số góc và giao điểm y (sắp xếp theo hệ số góc trước rồi theo y) cùng với một biến left thêm, x nhỏ nhất sao cho đường thẳng này đạt cực tiểu trong tập các đường thẳng. Sắp đường thẳng này vào vị trí đúng của nó và những đường bị loại sẽ là các đường liên tiếp kế bên nó. Chúng ta dùng các điều kiện tương tự trên để bỏ các đường thẳng bên trái và bên phải nó.Để trả lời truy vấn, chúng ta dùng một *set* nữa dùng chính các biến ấy nhưng lại sắp xếp theo left. Vậy mỗi lần truy vấn ta có thể dễ dàng chặt nhị phân để tìm ra kết quả như đã nói ở trên.

Với mục đích tổng hợp lại một số kiến thức cơ bản đề cập đến kỹ thuật baolồi đã được trình bày trước đây, do chưa có nhiều kinh nghiệm nên chuyên đề chắc chắn vẫn còn nhiều thiếu sót, rất mong nhận được sự góp ý của bạn đọc.Xin chân thành cảm ơn./.

Tài liệu tham khảo

[1] http://codeforces.com\

[2] http://vnoi.info/wiki

[3] https://www.giaithuatlaptrinh.com