Cải tiến quy hoạch động bằng bao lồi

MỤC LỤC

Giới thiệu

Mục đích của chuyên đề nhằm cải tiến thuật toán quy hoạch động với thời gian $O(n^2)$ xuống còn $O(n \log n)$ hoặc tốt hơn là xuống còn $O(n)$ cho một lớp các bài toán sử dụng kĩ thuật bao lồi - convex hull trick.

Bài toán 1.

Cho một bài toán với n phần tử khác nhau $1, 2, \dots, n$. Phần tử thứ i có hai tính chất $A[i], B[i]$. Hàm mục tiêu của bài toán này có thể được biểu diễn thông qua bảng quy hoạch động một chiều $C[1,2, \dots, n]$ thoả mãn hệ thức sau:

$$C[i] = \begin{cases} B[i] \text{ nếu } i = 1 \\ \min\limits_{j<i}\{C[j] + A[i]B[j]\}, \textit{trong trường hợp khác} \end{cases}$$

Tìm giá trị $C[n]$

Từ công thức quy hoạch động trên, ta có thể thiết kế giải thuật $O(n^2)$ để giải bài toán như sau:

```
function(A[1, 2, ..., n], B[1, 2, ..., n]);
C[1] ← B[1]
 for i ← 2 to n
tmp ← +∞
for j ← 1 to i − 1
tmp ← min(tmp, C[j] + A[i]B[j]
C[j] ← tmp

return C[n]
```

Mục tiêu tiếp theo: Cải tiến từ $O(n^2)$ xuống còn $O(n\ logn)$

Kĩ thuật bao lồi

Trước hết, xem xét bài toán hình học sau:

Bài toán 2.

Cho một tập n đường thẳng $D = \{d_1, d_2, \dots, d_n\}$ trong đó $(d_i)\ y_i = a_i x + b_i (a_i \geq 0)$ và k điểm trên trục Ox: $p_1, p_2, \dots, p_k$. Tìm $q_1, q_2, \dots, q_k$ trong đó mỗi giá trị q_ℓ được định nghĩa như sau: $q_\ell = \min\limits_{1 \leq i \leq n} a_i \cdot p_\ell + b_i$

Với mỗi điểm p_ℓ, bằng cách duyệt qua tất cả các đường thẳng trong D, ta có thể tính được q_ℓ trong thời gian $O(n)$. Do đó, tổng thời gian để tìm tất cả các điểm $q_1, a_2, \dots, q_k$ là O_{kn}. Tuy nhiên ta cũng có thể tìm các điểm này nhanh hơn dựa theo bổ đề sau:

Bổ đề 1.

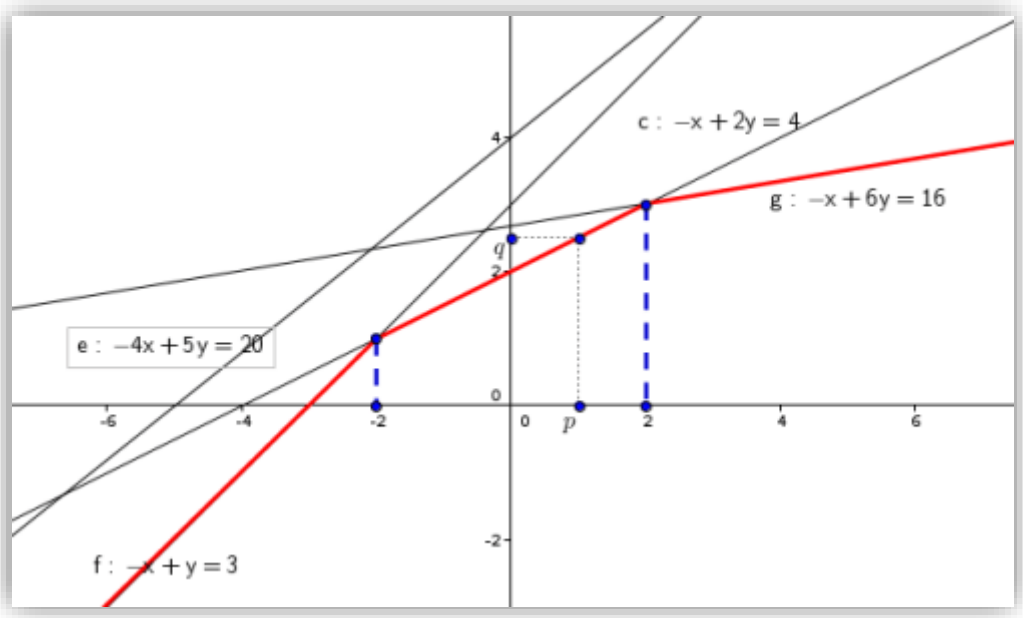
Tồn tại một thuật toán tìm tất cả các điểm $q_1, q_2, \dots, q_k$ trong thời gian $O((k+n)\log n)$.

Bổ đề 1 đúng cả với trường hợp tập các đường thẳng D có đường thẳng với $a_i < 0$. Tuy nhiên, trong các ứng dụng với bài toán quy hoạch động, các a_i luôn không âm. Ngoài ra, điều kiện $a_i \geq 0$ cũng làm cho việc phân tích thuật toán trở nên đơn giản hơn.

Trước hết chúng ta hãy xem ví dụ với $D = \{-x + y = 3, -x + 2y = 4, -x + 6y = 16, -4x + 5y = 20\}$.

Có thể thấy, với mỗi điểm $(p, 0) \in Ox$ giá trị nhỏ nhất tương ứng $(0, q)$ thoả mãn (p, q) thuộc phần gạch đỏ (hình trang 5). Phần gạch đỏ đó gọi là bao lồi (do đó kĩ thuật này còn gọi là kĩ thuật bao lồi). Như vậy, dựa vào hình vẽ trên, đường thẳng $-4x + 5y = 20$ trong D là dư thừa (tung độ của điểm có hoành độ p trên

đường thẳng này luôn lớn hơn tung độ của điểm có cùng hoành độ trên một đường thẳng khác của D).



Dựa vào ví dụ trên, ta thấy nếu chúng ta có một cách biểu diễn bao lồi phù hợp, chúng ta có thể nhanh chóng xác định được tung độ q nhỏ nhất ứng với hoành độ p.

Ta có thể biểu diễn bao lồi bằng tập các interval và một đường thẳng tương ứng mỗi interval. Với ví dụ trên, tập các interval là $I_1 = [-\infty, -2], I_2 = [-2, 2], I_3 = [2, +\infty]$ và các đường thẳng tương ứng lần lượt là $-x + y = 3, -x + 2y = 4, -x + 6y = 16$. Bổ đề 2 sau đây cho ta biết số interval trong biểu diễn bao lồi luôn nhỏ hơn hoặc bằng số đường thẳng.

Bổ đề 2

Bao lồi của tập các đường thẳng $D = \{d_1, d_2, \dots, d_n\}$ cho trước có thể biểu diễn bằng m interval $I_1, I_2, \dots, I_m$ và m đường thẳng tương ứng với mỗi đoạn $d_1, d_2, \dots, d_m$ với $m \leq n$. Hơn nữa, ta có thể tìm được biểu diễn đó trong thời gian $O(nlogn)$.

Để đơn giản, ta giả sử trong D không có hai đường thẳng nào song song. Nếu có hai đường thẳng song song, ta chỉ cần thay đổi một chút trong giả mã dưới đây mà không thay đổi thời gian tính của thuật toán. Thuật toán tìm bao lồi của D như sau :

$\underline{FindHull(d_1, d_2, \dots, d_n)}$:
  $sort\ \{d_1, d_2, \dots, d_n\}\ by\ decreasing\ order\ of\ slope$ $Stack\ S \leftarrow \emptyset$
    $S.push(d_1)$
    $I_1 \leftarrow [-\infty, +\infty]$
    $m \leftarrow 1$
    for $i \leftarrow 2\ to\ n$
      $d \leftarrow S.peek()$
      $x_p \leftarrow\ FindIntersectionX(d, d_i)$
      while $x_p \leq left(I_m)$
        $S.pop()$
        $m \leftarrow m - 1$
        $d \leftarrow S.peek()$
        $x_p \leftarrow\ FindIntersectionX(d_i, d)$
      $S.push(d_i)$
      $I_m \leftarrow [left(I_m), x_p]$
      $I_{m+1} \leftarrow [x_p, +\infty]$
      $associate\ I_{m+1}\ with\ d_i$
    $m \leftarrow m + 1$

Full Code

```
typedef struct{
    double a;
    double b;
  } double_pair;

 struct Stack_node {
    int lineId;
    struct Stack_node *next;
  };
typedef struct Stack_node stack_node;

stack_node *S;                // the stack
double_pair D[MAX_SIZE];      // the set of lines
double_pair I[MAX_SIZE];      // the set of intervals
double      Q[MAX_SIZE];      // the set of query points
```

```
int         ALines[MAX_SIZE];// the set of lines associated
with intervals

int find_hull(int n){
    qsort(D, n, sizeof(*D), slope_compare);
    S = malloc( sizeof(stack_node));
    S->lineId = -1; // head of the stack
    push(0);        // push d_1 to the stack;
    I[0].a = -(double)INFTY;
    I[0].b = (double)INFTY;
    int m = 0;
    ALines[m] = 0;
    int i = 0;
    stack_node *d;
    double x = 0.0;
    for(i = 1; i <n ;i++){
        d = peek();
        x = find_intersection_x(d->lineId, i);
        while(x <= I[m].a){  // found a redundant line
            pop();       // remove the redundant line
            m--;
            d = peek();
            x = find_intersection_x(d->lineId, i);
        }
        push(i);             // push d_i to the stack
        I[m].b = x;
        I[m+1].a = x;
        I[m+1].b = (double)INFTY;
        ALines[m+1] = i;
        m++;
    }

    return m+1;        // the number of intervals
}

double find_intersection_x(int x, int y){
    double_pair d_a = D[x];
    double_pair d_b = D[y];
    return (d_b.b - d_a.b)/(d_a.a - d_b.a);
}
```

Thủ tục $FindIntersectionX(d_1, d_2)$ tìm hoành độ giao điểm của hai đường thẳng d_1, d_2 (chi tiết coi như bài tập cho bạn đọc).

Ta có thể thấy vòng for của thuật toán tìm bao lồi có thời gian chạy $O(n)$ vì mỗi đường thẳng sẽ được xem xét tối đa hai lần: khi đưa vào $stack$ và khi lấy ra khỏi $stack$.

Nếu một đường thẳng bị lấy ra khỏi $stack$, nó sẽ không bao giờ được kiểm tra lại nữa. Do đó, thời gian của thuật toán tìm bao lồi chủ yếu dành để sắp xếp các đường thẳng theo slope và mất $O(nlogn)$.

Chứng minh Bổ đề 1

Giả sử chúng ta đã tìm được bao lồi của D, với mỗi điểm p_ℓ trên trục hoành, sử dụng tìm kiếm nhị phân để tìm interval $I \in I_1, I_2, \dots, I_m$ sao cho $p_\ell \in I$.

Giả sử $(d)y = ax + b$ là đường thẳng tương ứng với i, ta sẽ có $q_\ell = a \cdot p_\ell + b$.

Do đó, ta có thể tính q_ℓ trong thời gian $O(logn)$. Nếu ta có k điểm $p_1, p_2, \dots, p_k$, ta có thể tìm $q_1, q_2, \dots, q_k$ trong thời gian $O(klogn)$.

Chi tiết thuật toán như sau:

```
Query({d_1, d_2, ..., d_n}, {p_1, p_2, ..., p_k}):
  FindHull(d_1, d_2, ..., d_n)
  for i ← 1 to k
    I ← IntervalBinSearch({I_1, I_2, ..., I_m}, p_i)
    d ← the line associated with I
    output q_i = a · p_i + b          [[assuming (d)y = ax + b]]
```

else if $p > right(I\ell)$
  $return\ IntervalBinSearch(\{I\ell + 1, \dots, Im\}, p)$
else
  $return\ IntervalBinSearch(\{I1, \dots, I\ell - 1\}, p)$

CODE

```
void query(double points[], int k, int n){
    int m = find_hull(n);
    int i = 0, q = 0;
    for(i = 0; i < k; i++){
        q = interval_search(0, m-1, points[i]);
        Q[i] = D[ALines[q]].a*points[i] + D[ALines[q]].b;
    }
}


int interval_search(int x, int y, double p){
    if(y <= x){
        return x;
    } else {
        int mid = (x+y)/2;
        if((I[mid].a <= p) && (p <= I[mid].b)){
            return mid;
        } else if (p > I[mid].b){
            return interval_search(mid+1, y, p);
        } else {
            return interval_search(x, mid-1, p);
        }
    }
}
```

Quay trở lại bài toán quy hoạch động chúng ta đang xét. Nếu ta đặt $(d_j)y_j = B[j]x + C[j], 1 \leq j \leq n$ thì $C[i]$ chính là tung độ q nhỏ nhất thuộc một trong các đường thẳng $d_1, d_2, \dots, d_{i-1}$ tương ứng với hoành độ $p = A[i]$.

Do đó, chúng ta có thể áp dụng ý tưởng của kĩ thuật bao lồi vào bài toán quy hoạch động này.

Trường hợp đặc biệt: Thuật toán $O(nlogn)$

Trước tiên chúng ta xét trường hợp mảng $B[1,2,\dots,n]$ thỏa mãn tính chất sau:

$$B[1] \geq B[2] \geq \cdots \geq B[n]$$

Giả sử $B[1] \geq B[2] \geq \cdots \geq B[n]$ tuy khá mạnh nhưng rất nhiêu bài toán chúng ta gặp sẽ thảo mãn điều kiện này. Với giả sử này, slope cuả các đường thẳng $d_1, d_2, \dots, d_n$ đã được sắp xếp theo thứ tự giảm dần. Do đó ta tiết kiệm được bước sắp xếp trong thuật tóan $FindHull(d1, d2, \dots, dn)$.

Để giả mã đơn giản, ta giả sử trong $B[i] \neq B[i+1]$ với mọi i. Nếu tồn tại i sao cho $B[i] = B[i+1]$, ta chỉ cần thay đổi một chút trong giả mã dưới đây mà không thay đổi thời gian tính của thuật toán. Thuật toán chi tiết như sau :

```
FastDynamic(A[1,2,...,n], B[1,2,...,n]):
  d1 ← B[1]x + C[1]
  Stack S ← ∅
  S.push(d1)
  I_1 ← [−∞, +∞]
  m ← 1
  for i ← 2 to n
    I ← IntervalBinSearch({I_1, I_2, ..., I_m}, A[i])
    d ← the line associated with I
    C[i] ← a · A[i] + b          [[assuming (d)y = ax + b]]
    di ← B[i]x + C[i]
    d ← S.peek()                 [[examine the top element]]
    xp ← FindIntersectionX(d, d_i)
    while xp ≤ left(I_m)         [[found a redundant line]]
      S.pop()
      m ← m − 1
      d ← S.peek()
      xp ← FindIntersectionX(d_i, d)
    S.push(d_i)
    I_m ← [left(I_m), x_p]
    I_m + 1 ← [x_p, +∞]
    associate I_{m+1} with d_i
    m ← m + 1
  return C[n]
```

Trường hợp tổng quát

Trong trường hợp tổng quát, chúng ta có thể áp dụng thủ tục $IntervalBinSearch(\{I_1, I_2, \dots, I_m\}, A[i])$ để tìm kiếm interval chứa $A[i]$.
Do đó chúng ta mất $O(logn)$ cho việc tìm kiếm interval cho mỗi vòng lặp của thuật toán.
Vấn đề còn lại chỉ là làm sao để cập nhật bao lồi trong mỗi bước khi ta thêm đường thẳng $(d)B[i]x + C[i]$.
Ở đây chỉ đưa ra ý tưởng để thực hiện cập nhật bao lồi trong thời gian trung bình $O(logn)$ mỗi bước. Do đó, tổng thời gian của thuật toán là $O(nlogn)$.

Ý tưởng của thuật toán dựa trên thuật toán Sweep Line. Ta sẽ luôn luôn lưu các đường thẳng trong bao lồi theo thứ tự tăng dần của slope sau mỗi bước. Gọi $d_1, d_2, \dots, d_m$ là các đường thẳng trong bao lồi sau bước $i - 1$.

Để đơn giản ta giả sử không có hai đường thẳng nào song song. Trong trường hợp có hai đường song song, ta chỉ cần thay đổi thuật toán một chút.

Bây giờ ta có thêm đường thẳng $(di)y = a_i x + b_i$, để cập nhật bao lồi trước hết ta thực hiện tìm kiếm nhị phân để tìm ra đường thẳng $(dj)y = a_j y + b_j$ sao cho $a_j > a_i > a_j + 1$.

Trường hợp $j = m$, ta cập nhật như trong trường hợp đặc biệt. Do đó, ta có thể giả sử $j < m$.

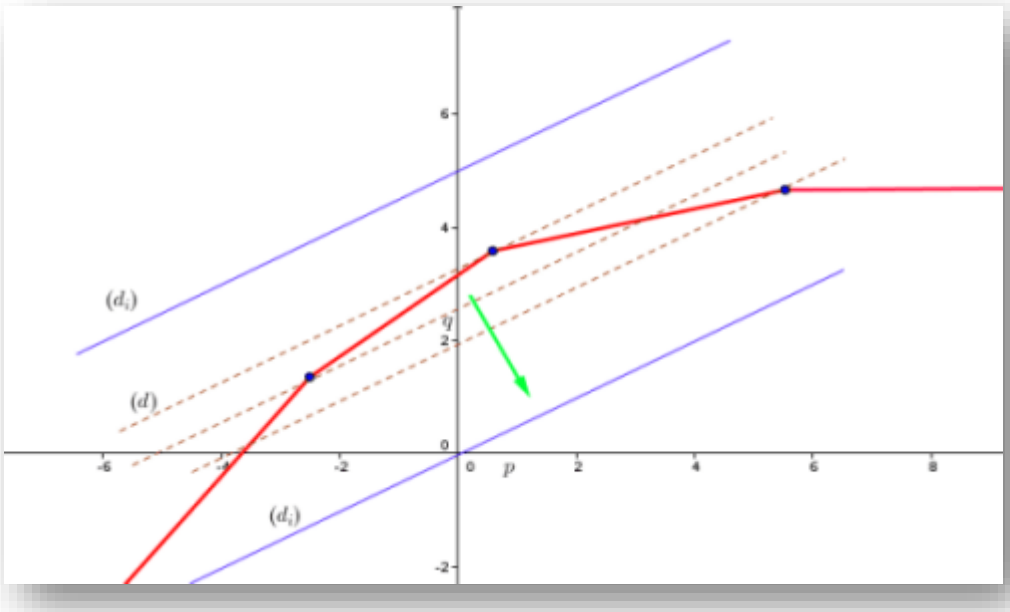
Gọi (x_p, y_p) là giao điểm của d_j và d_{j+1}.

Xét đường thẳng $(d)y = a_i x + b_i'$ có cùng slope a_i với d_i và đi qua (x_p, y_p). Nếu $b_i' \leq b_i$ (đường thẳng d_i có màu xanh trong hình tại trang 12), đường thẳng d_i là dư thừa và do đó, bao lồi không thay đổi khi ta thêm đường thẳng d_i.

Nếu $b_i' > b_i$, ta lần lượt "dịch" đường thẳng d về phía d_i cho đến khi d trùng với d_i.

Nếu có sự "thay đổi" của bao lồi (các đường thẳng màu nâu), ta cập nhật lại bao lồi.

Ta có thể nhận thấy trong trường hợp tồi nhất, ta phải dịch đường thẳng d $O(m)$ lần. Tuy nhiên, nếu ta dịch K lần, thì mỗi lần dịch ta sẽ "loại bớt" được K đường thẳng dư thừa và các đường thẳng này sẽ không được xét đến trong các vòng lặp tiếp theo. Do đó, thời gian trung bình mỗi bước lặp vẫn là $O(logn)$



Ví dụ áp dụng

Bài 1. Kalila and Dimna in the Logging Industry (http://codeforces.com/contest/319/problem/C)

Tóm tắt đề bài: Có n cái cây $1,2,\dots,n$ trong đó cây thứ i có chiều dài $A[i] \in N$. Bạn phải cắt tất cả các cây thành các đoạn có chiều dài 1.

Bạn có một cái cưa máy (lởm), mỗi lần chỉ chặt được một đơn vị chiều dài, và mỗi lần sử dụng thì lại phải sạc pin. Chi phí để sạc pin phụ thuộc vào chi phí của cây đã bị cắt hoàn toàn (một cây bị cắt hoàn toàn có chiều dài 0).

Nếu chỉ số lớn nhất của cây bị cắt hoàn toàn là i thì chi phí sạc là $B[i]$, và khi bạn đã chọn một cây để cắt, bạn phải cắt nó hoàn toàn.

Ban đầu cái cưa máy được sạc đầy pin.

Giả sử $a_1 = 1 \le a_2 \le \cdots \le a_n$ và $b_1 \ge b_2 \ge \cdots \ge b_n = 0$.

Tìm một cách cưa cây với chi phí nhỏ nhất.

Hướng dẫn giải thuật

Ví dụ: $n = 6, A[1,2,\dots,6] = \{1,2,3,10,20,30\}, B[1,2,\dots,6] = \{6,5,4,3,2,0\}$
Nếu bạn chặt cây theo thứ tự $1 \to 2 \to 6 \to 3 \to 4 \to 5$ chi phí bạn phải trả là
$2 \times 6 + 30 \times 5 + 3 \times 0 + 10 \times 0 + 20 \times 0 = 162.$
Nếu bạn chặt theo thứ tự: $1 \to 3 \to 6 \to 2 \to 4 \to 5$ chí phí bạn phải trả là
$3 \times 6 + 30 \times 4 + 4 \times 0 + 10 \times 0 + 20 \times 0 = 138$

Từ ví dụ trên, ta có nhận xét sau: Chi phí để chặt các cây sau khi đã chặt cây thứ n là 0.

Do đó, bài toán trên có thể được quy về bài toán: tìm chi phí nhỏ nhất để chặt cây thứ n.

Gọi $OPT = \{1 = i_1, i_2, \dots, i_k\}$ là một thứ tự chặt cây tối ưu với $i_k = n$. Ta có:

$$1 = i_1 \le i_2 \le \cdots \le i_k = n$$

Phát triển quy hoạch động như sau:

Gọi $C[i]$ là chi phí nhỏ nhất để chặt cây thứ i "sử dụng" các cây $1, 2, \dots, i-1$. Ta có công thức sau:

$$C[i] = \begin{cases} B[i] \text{ nếu } i = 1 \\ \min\limits_{j<i}\{C[j] + A[i]B[j] + B[j], \textit{trong trường hợp khác} \end{cases}$$

Dễ thấy, chi phí để chặt cây thứ n là $C[n]$.

Để ý kĩ sẽ thấy công thức quy hoạch động để tính $C[n]$ với hai mảng A, B là các mảng đã sắp xếp. Do đó, có thể giải bài toán này trong thời gian $O(n)$.

Code xem trang 14

```
#include<stdio.h>
#include<iostream>
#include<stdlib.h>
#include<math.h>
#include<stack>

#define min(x, y) (((x) < (y)) ? (x) : (y))
#define max(x, y) (((x) < (y)) ? (y) : (x))
#define is_equal(x,y) (((x) - (y) == 0) ? 1 : 0)
#define MAX_SIZE 100005
#define INFTY  1e15 + 7
#define NULL 0

using namespace std;

typedef struct{
    double a;
    double b;
  } double_pair;


stack<int> S;
double_pair I[MAX_SIZE];
long long           A[MAX_SIZE];
long long      B[MAX_SIZE];
long long C[MAX_SIZE];
int         ALines[MAX_SIZE];// the set of lines associated with
I

double find_intersection_x(int x, int y);
void readinput();
void fast_fats_dynamic(long long A[], long long B[],int n);
int interval_search(int x, int y, double p);
int n;



void readinput(){
     cin >> n;
     for(int i = 0 ; i < n; i++) cin>>A[i];
     for(int i = 0 ; i < n; i++) cin>>B[i];
}

void fast_fast_dynamic(long long A[], long long B[], int n){
     C[0] = B[0]; // d_1 = B[0]x + C[0]
```

```
     S.push(0); // push d_1 to the top of the stack;
     I[0].a = -(double)INFTY;
     I[0].b = (double)INFTY;
     int m = 0;
     ALines[m] = 0;
     int i = 0, q = 0, l =0;
     int d;
     double x = 0.0;
     int prev_interval = 0;
     for(i = 1; i <n ;i++){
        l = prev_interval-1; // the interval associated with A[i-
1]
        while(1){
          l++;
          if((I[l].a   <=   (double)A[i])   &&   ((double)A[i]<=
I[l].b)){
               q = l;
               break;
          }
        }

          C[i] = B[ALines[q]]*(A[i]-1) + C[ALines[q]] + B[i];
          d = S.top();   // examine the top element of the stack
          x = find_intersection_x(d, i);
          while(x <= I[m].a){  // found a redundant line
               S.pop();           // remove the redundant line
               m--;
               d = S.top();
               x = find_intersection_x(d, i);
          }
          prev_interval = l;
        if(x < (double) (INFTY-1)){
            S.push(i);
            I[m].b = x;
            I[m+1].a = x;
            I[m+1].b = INFTY;
            if( l >= m){
                 if    ((I[m].a    <=    ((double)A[i]))    &&
(((double)A[i]) <= I[m].b)){
                    prev_interval = m;
                }else {
                     prev_interval = m+1;
                }
            }
            ALines[m+1] = i;
```

```
                m++;
            }
        }
cout<<C[n-1]<<endl;
}

int interval_search(int x, int y, double p){
     if(y <= x){
          return x;
     } else {
          int mid = (x+y)/2;
          if((I[mid].a <= p) && (p <= I[mid].b)){
               return mid;
          } else if (p > I[mid].b){
               return interval_search(mid+1, y, p);
          } else {
               return interval_search(x, mid-1, p);
          }
     }
}
double find_intersection_x(int x, int y){
     long long a1 = B[x], b1 = C[x];
    long long a2 = B[y], b2 = C[y];
    return ((double)(b2-b1))/((double)(a1 - a2));
}


int main(int argc, const char * argv[]){
     readinput();
     fast_fast_dynamic(A,B,n);
     return 0;
}
```

Bài 2. 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).

Hướng dẫn giải thuật bằng bao lồi

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 bài toán trở về tìm hàm cực tiểu của $y = m_j x + b_j$ bằng cách tìm j tối ưu.

```
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.

Bài 3. Phân nhóm

Cho $n \leq 300000$ cặp số (x, y) $(1 \leq x, y \leq 1000000)$. Ta có thể nhóm một vài cặp số lại thành một nhóm.
Giả sử một nhóm gồm các cặp số thứ $a_1, a_2, \dots, a_m$ thì chi phí cho nhóm này sẽ là $max(x_{a_1}, x_{a_2}, \dots, x_{a_m}) * max(y_{a_1}, y_{a_2}, \dots, y_{a_m}$.
Yêu cầu: tìm cách phân nhóm có tổng chi phí bé nhất.
Input

- Dòng đầu tiên là số nguyên dương N.
- N dòng tiếp theo dòng thứ i gồm hai số xi và yi.

Output

- Gồm 1 số duy nhất là kết quả tìm được.

Ví dụ
Input
4
100 1
15 15
20 5
1 100

Output
500
Giải thích: cách phân nhóm thích hợp là (1), (2,3) và (4).

Hướng dẫn giải thuật.

Nhận xét 1

Nếu tồn tại hai cặp số $(x1, y1)$ và $(x2, y2)$ mà $x1 > x2$ và $y1 > y2$ thì ta nói cặp số $(x2, y2)$ là không tiềm năng, và có thể loại bỏ không cần quan tâm tới nó. Bởi vì ta có thể cho nó vào cùng nhóm với cặp đầu tiên mà không làm tồi đi kết quả.

Như vậy ta có thể sắp xếp lại các cặp số tăng dần theo x . Sau đó sử dụng stack để loại bỏ đi những cặp số không tiềm năng, cuối cùng còn lại một dãy các cặp với x tăng dần và y giảm dần. Từ đây ta chỉ cần giải bài toán cho dãy các cặp số này.

Nhận xét 2

Nếu ta có hai cặp số $(x1, y1)$ và $(x2, y2)$ thuộc cùng một nhóm, thì ta có thể thêm vào nhóm đấy các cặp số (x, y) mà $x1 \leq x \leq x2$ và $y1 \geq y \geq y2$ mà không làm tồi đi kết quả. Như vậy có nhận xét: các nhóm được phân hoạch gồm các phần tử liên tiếp nhau.

Quy hoạch động

Với hai nhận xét trên, ta đã có thể có được thuật toán quy hoạch động đơn giản với độ phức tạp đa thức. Gọi $F(i)$ là chi phí nhỏ nhất để phân nhóm các phần tử có chỉ số không quá I .

Công thức truy hồi: $F(i) = min[F(j) + xi * yj]$ $với$ $0 \leq j < i$

Có thể cài đặt thuật toán trên với độ phức tạp $O(N^2)$, tuy nhiên như vậy vẫn chưa đủ tốt với giới hạn của đề bài.

Áp dụng bao lồi

Đặt $yj = a$, $xi = x$, và $F(j) = b$. Rõ ràng ta cần cực tiểu hóa một hàm bậc nhất $y = a * x + b$ bằng việc chọn j hợp lí. Đồng thời trong bài toán này thì hệ số góc a của các đường thẳng là giảm dần, như vậy có thể áp dụng trực tiếp bao lồi.

Để ý là các truy vấn (các giá trị x) là tăng dần, nên ta không cần phải tìm kiếm nhị phân mà có thể tịnh tiến để tìm kết quả. Độ phức tạp cho phần quy hoạch động này là $O(N)$.

```
#include <bits/stdc++.h>
#define X first
#define Y second

const int N = 300005;
```

```
using namespace std;
typedef pair<long long, long long> Line;

int n;
pair<int, int> a[N];

long long eval(long long x, Line line) {
    return x * line.X + line.Y;
}

bool bad(Line d1, Line d2, Line d3) {
    return (d2.Y - d1.Y) * (d1.X - d3.X) >= (d3.Y - d1.Y) *
(d1.X - d2.X);
}

int main() {
    scanf("%d", &n);
    int i;
    for (i = 1; i <= n; i++) scanf("%d %d", &a[i].X, &a[i].Y);
    sort(a + 1, a + 1 + n);
    vector<pair<int, int> > b;
    for (i = 1; i <= n; i++) {
        while (b.size() && b.back().Y < a[i].Y) b.pop_back();
        b.push_back(a[i]);
    }
    vector<Line> d; long long last = 0; Line new_line; int best
= 0;
    for (i = 0; i < b.size(); i++) {
        new_line = Line(b[i].Y, last);
        while (d.size() >= 2 && bad(d[d.size() - 2], d[d.size()
- 1], new_line)) {
            if (best >= d.size() - 1) best--; d.pop_back();
        }
        d.push_back(new_line);
        while (best + 1 < d.size() &&
          eval(b[i].X, d[best]) >= eval(b[i].X, d[best + 1]))
best++;
        last = intersectX(b[i].X, d[best]);
    }
    cout << last << endl;
    return 0;
}
```

Một số bài tập tự luyện

1. http://lequydon.ntucoder.net/Problem/Details/4612
2. https://open.kattis.com/problems/joiningnetwork
3. https://vn.spoj.com/problems/RAOVUON/

Tài liệu tham khảo

1. http://codeforces.com/blog/entry/8219
2. http://www.giaithuatlaptrinh.com/?p=176
3. http://wcipeg.com/wiki/Convex_hull_trick
4. https://vnoi.info/wiki/translate/wcipeg/Convex-Hull-Trick
5. http://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong