Cải tiến quy hoạch động bằng bao lồi

MỤC LỤC

[Giới thiệu 2](#_Toc15565814)

[Bài toán 1. 3](#_Toc15565815)

[Kĩ thuật bao lồi 4](#_Toc15565816)

[Bài toán 2. 4](#_Toc15565817)

[Bổ đề 1. 4](#_Toc15565818)

[Bổ đề 2 5](#_Toc15565819)

Giới thiệu

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

## Bài toán 1.

Cho một bài toán với phần tử khác nhau . Phần tử thứ có hai tính chất . Hàm mục tiêu của bài toán này có thể được biểu diễn thông qua bảng quy hoạch động một chiều thoả mãn hệ thức sau:

Tìm giá trị

Từ công thức quy hoạch động trên, ta có thể thiết kế giải thuật để giải bài toán như sau:

|  |
| --- |
| **function;**    **for to**    **for**      **return** |

Mục tiêu tiếp theo: Cải tiến từ xuống còn

## Kĩ thuật bao lồi

Trước hết, xem xét bài toán hình học sau:

### Bài toán 2.

Cho một tập đường thẳng trong đó và điểm trên trục . Tìm trong đó mỗi giá trị được định nghĩa như sau:

Với mỗi điểm , bằng cách duyệt qua tất cả các đường thẳng trong , ta có thể tính được trong thời gian . Do đó, tổng thời gian để tìm tất cả các điểm là . Tuy nhiên ta cũng có thể tìm các điểm này nhanh hơn dựa theo bổ đề sau:

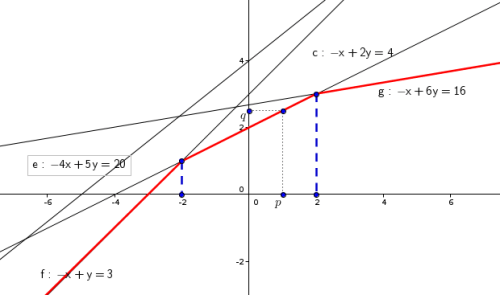
### Bổ đề 1.

Tồn tại một thuật toán tìm tất cả các điểm trong thời gian .

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

Trước hết chúng ta hãy xem ví dụ với .

Có thể thấy, với mỗi điểm giá trị nhỏ nhất tương ứng thoả mãn 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 trong là dư thừa (tung độ của điểm có hoành độ trên đường thằng này luôn lớn hơn tung độ của điểm có cùng hoành độ trên một đường thẳng khác của ).



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 nhỏ nhất ứng với hoành độ .

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 và các đường thẳng tương ứng lần lượt là . 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 cho trước có thể biểu diễn bằng interval và đường thẳng tương ứng với mỗi đoạn với . Hơn nữa, ta có thể tìm được biểu diễn đó trong thời gian .

Để đơn giản, ta giả sử trong 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 như sau :

|  |
| --- |
|  |

|  |
| --- |
| 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 tìm hoành độ giao điểm của hai đường thẳng (chi tiết coi như bài tập cho bạn đọc).

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

Nếu một đường thẳng bị lấy ra khỏi , 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 .

**Chứng minh Bổ đề 1**

Giả sử chúng ta đã tìm được bao lồi của , với mỗi điểm trên trục hoành, sử dụng tìm kiếm nhị phân để tìm interval sao cho .

Giả sử là đường thẳng tương ứng với , ta sẽ có .

Do đó, ta có thể tính trong thời gian . Nếu ta có điểm , ta có thể tìm trong thời gian .

Chi tiết thuật toán như sau:

|  |
| --- |
|  |
|  |
|  |
| 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 thì chính là tung độ nhỏ nhất thuộc một trong các đường thẳng tương ứng với hoành độ .

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 thỏa mãn tính chất sau:

Giả sử 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 đã được sắp xếp theo thứ tự giảm dần. Do đó ta tiết kiệm được bước sắp xếp trong thuật tóan

Để giả mã đơn giản, ta giả sử trong với mọi . Nếu tồn tại sao cho , 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 :

|  |
| --- |
|  |

### Trường hợp tổng quát

Trong trường hợp tổng quát, chúng ta có thể áp dụng thủ tục để tìm kiếm interval chứa .

Do đó chúng ta mất 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 thằng .

Ở đây chỉ đưa ra ý tưởng để thực hiện cập nhật bao lồi trong thời gian trung bình mỗi bước. Do đó, tổng thời gian của thuật toán là .

Ý tưởng của thuật toán dựa trên thuật toán [Sweep Line](https://en.wikipedia.org/wiki/Sweep_line_algorithm). Ta sẽ luôn luôn lưu các đường thẳng trong bao lồi theo thứ tự tăng dần của slope sau mỗi bước. Gọi là các đường thẳng trong bao lồi sau bước .

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 thằng , để cập nhật bao lồi trước hết ta thực hiện tìm kiếm nhị phân để tìm ra đường thẳng sao cho .

Trường hợp , ta cập nhật như trong trường hợp đặc biệt. Do đó, ta có thể giả sử .

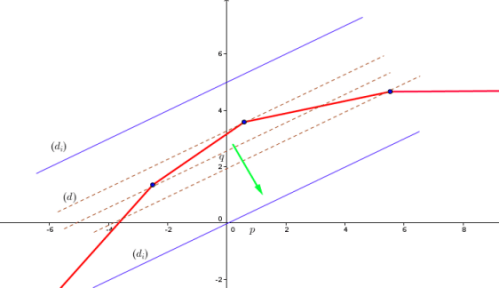
Gọi là giao điểm của và .

Xét đường thằng có cùng slope với và đi qua . Nếu (đường thằng có màu xanh trong hình tại trang 12), đường thằng là dư thừa và do đó, bao lồi không thay đổi khi ta thêm đường thằng .

Nếu , ta lần lượt "dịch" đường thằng về phía cho đến khi trùng với .

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 thằng lần. Tuy nhiên, nếu ta dịch lần, thì mỗi lần dịch ta sẽ "loại bớt" được 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



## 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 cái cây trong đó cây thứ có chiều dài . Bạn phải cắt tất cả các cây thành các đoạn có chiều dài .

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 ).

Nếu chỉ số lớn nhất của cây bị cắt hoàn toàn là thì chi phí sạc là , 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 và .

Tìm một cách cưa cây với chi phí nhỏ nhất.

Hướng dẫn giải thuật

Ví dụ:

Nếu bạn chặt cây theo thứ tự chi phí bạn phải trả là .

Nếu bạn chặt theo thứ tự: chí phí bạn phải trả là

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

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

Gọi là một thứ tự chặt cây tối ưu với . Ta có:

Phát triển quy hoạch động như sau:

Gọi là chi phí nhỏ nhất để chặt cây thứ "sử dụng" các cây . Ta có công thức sau:

Dễ thấy, chi phí để chặt cây thứ là .

Để ý kĩ sẽ thấy công thức quy hoạch động để tính với hai mảng là các mảng đã sắp xếp. Do đó, có thể giải bài toán này trong thời gian .

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 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 với là chiều rộng của hình chữ nhật và là chiều dài của hình chữ nhật .

Vậy bài toán trở về tìm hàm cực tiểu của bằng cách tìm 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 , ta có thể dễ dàng thấy thời gian thực hiện của cả bài toán là . 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 nhưng tổng độ phức tạp vẫn là do chi phí sắp xếp.

### Bài 3. Phân nhóm

Cho cặp số . Ta có thể nhóm một vài cặp số lại thành một nhóm.

Giả sử một nhóm gồm các cặp số thứ thì chi phí cho nhóm này sẽ là .

**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 và mà và thì ta nói cặp số 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 . Sau đó sử dụng stack để loại bỏ đi những cặp số không tiềm năng, cuối cùng còn lại một dãy các cặp với tăng dần và 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 và thuộc cùng một nhóm, thì ta có thể thêm vào nhóm đấy các cặp số mà và 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 là chi phí nhỏ nhất để phân nhóm các phần tử có chỉ số không quá .

Công thức truy hồi:

Có thể cài đặt thuật toán trên với độ phức tạp , tuy nhiên như vậy vẫn chưa đủ tốt với giới hạn của đề bài.

**Áp dụng bao lồi**

Đặt và . Rõ ràng ta cần cực tiểu hóa một hàm bậc nhất bằng việc chọn hợp lí. Đồng thời trong bài toán này thì hệ số góc 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 ) 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 .

|  |
| --- |
| #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>