Máy chấm, bài toán hay với test đẹp

Trần Việt Khoa, Hồ Văn Giáo
tvkhoa.husc@gmail.com,
Khoa Công nghệ thông tin, Đại học Khoa học Huế

Ngày 16 tháng 3 năm 2017

1 Máy chấm

Thi lập trình trực tuyến OnlineJudge là một hình thức tổ chức thi, giải bài và chấm điểm mã nguồn trực tuyến đã trở nên phổ biến. Có thể đơn cử ra đây website: http://codeforces.com. Hình thức thi này đem lại rất nhiều lợi ích, lớn nhất có thể nêu là cung cấp một công cụ để dễ dàng tổ chức những kỳ thi lập trình với quy mô lớn, công bằng, minh bạch, tiết kiệm chi phí và thời gian. Tạo ra một môi trường học tập, nghiên cứu và rèn luyện khả năng tư duy lập trình của học sinh – sinh viên cùng những người đam mê.

Ngoài ra, nó còn là một sân chơi bổ ích, thú vị, một diễn đàn để giao lưu, kết nối và thử sức mình. Vì vậy sự cạnh tranh, ganh đua là sản phẩm tất yếu của môi trường kết nối đó. Là động lực thúc đẩy và tạo điều kiện tích cực cho sự phát triển của thuật toán, kỹ thuật lập trình hay tổng quan hơn là ngành công nghệ thông tin thế giới.

Quá trình chấm điểm được thực hiện hoàn toàn khách quan trên các bộ test được xây dựng từ trước, không bị ảnh hưởng bởi ý kiến chủ quan hay tâm lý người chấm.

Vậy bộ test là gì? nó được tạo ra như thế nào, nó có ảnh hưởng đến tư duy thuật toán của thí sinh hay không?. Rất nhiều câu hỏi liên quan đến đối tượng này và theo quan điểm của tác giả đây chính là điểm khó khăn nhất cho hệ thống thi trực tuyến, đòi hỏi người ra đề có kinh nghiệm, kiến thức chuyên môn cao.

Bài viết sau [2] cho bạn một cái nhìn về công việc này và đồng thời cũng là một kỹ thuật cao cấp dành cho thí sinh trong quá trình luyện tập, thi lập trình.

2 Bài toán hay, test đẹp và sự kế thừa

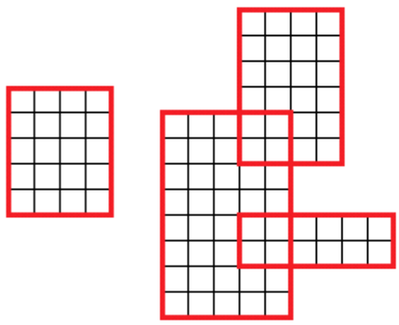
2.1 Bài toán tính diện tích hình chữ nhật

Đây là một bài toán "Mars Maps" trong kỳ thi BOI (Baltic Olympiad in Informatics) năm 2001, tham khảo đề [1]. Một bài toán rất đẹp về mặt ý tưởng, có nghĩa nó có thể được tiếp cận giải bằng nhiều phương pháp khác nhau và có thể tìm ra được một phương pháp hữu hiệu nhất. Từ bài toán này, một cấu trúc dữ liệu được đưa ra cho một lớp các bài toán tương tự đó là cấu trúc cây phân đoạn (Segment Tree). Ở đây ta không đề cập đến lịch sử sự ra đời của cấu trúc dữ liệu này, nhưng theo quan sát có lẻ bài toán này gần như xuất hiện sớm nhất và sau đó các bài toán tương tự được đề

xuất và dần hình thành một số kỹ thuật tối ưu khi xử lý cho bài toán dạng này, ví dụ kỹ thuật "Lazy propagation".

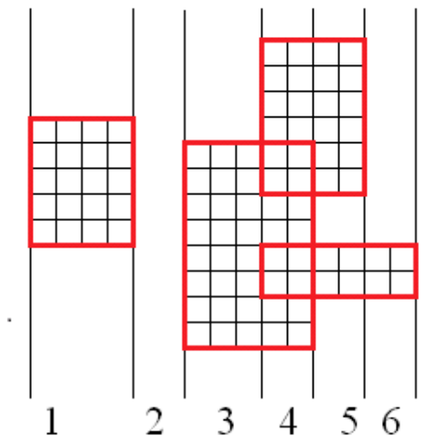
Trở lại với bài toán này, phần tiếp theo ta nghiên cứu lời giải của tác giả bài toán được biên dịch bởi Hồ Văn Giáo.

Xét bài toán với bộ dữ liệu gồm các hình chữ nhật như hình vẽ sau:



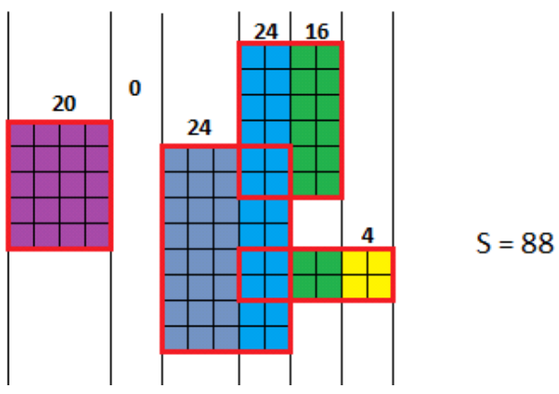
Hình 1. Dữ liệu.

Ta sẽ quét mặt phẳng từ trái sang phải và chia mặt phẳng thành các vùng bởi các cạnh dọc của các hình chữ nhật. Ta sẽ có 2 loại cạnh là cạnh mở đầu hình chữ nhật và cạnh đống hình. Như vậy ta sẽ có 6 vùng như hình dưới đây.



Hình 2. 6 vùng.

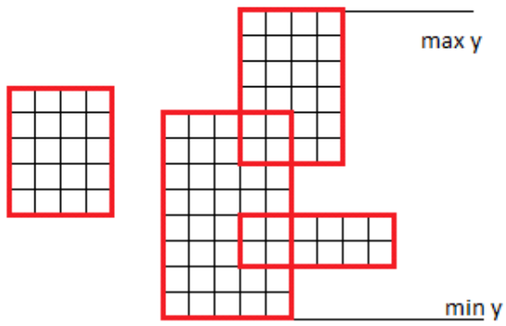
Vậy các vùng sẽ chứa những hình chữ nhật không chồng lên nhau. Suy ra diện tích các hình chữ nhật là tổng diện tích của các hình chữ nhật con trong các vùng đã chia, hình 3.



Hình 3. Các hình chữ nhật con và diện tích của nó.

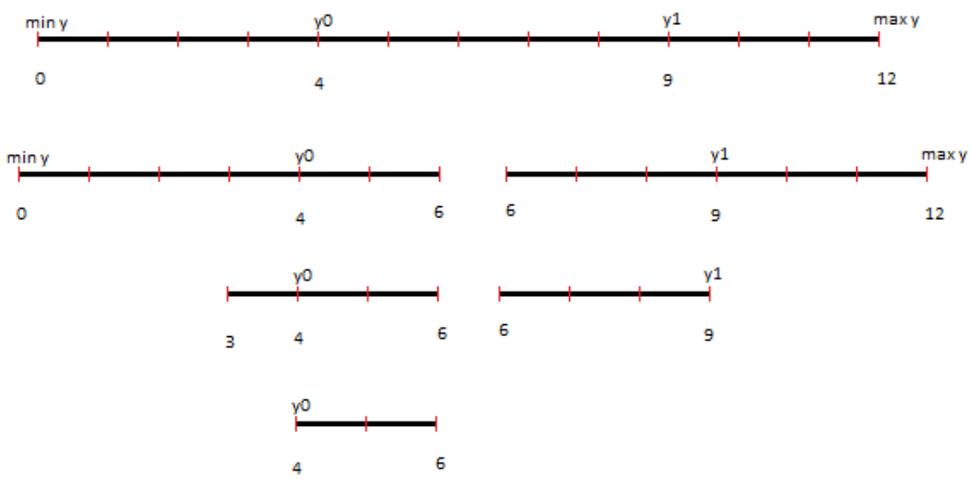
Vậy để tính được diện tích của các hình chữ nhật trong mỗi vùng thì ta cần biết tổng chiều cao các hình trong vùng đó và độ dài của vùng. Để chia vùng ta cần các cạnh dọc của các hình chữ nhật và để việc quét thuận tiện các cạnh cần được sắp xếp theo trật tự tăng dần của x. Độ dài của vùng được tích bằng vị trí x của cạnh hiện tại hiệu cho vị trí x của cạnh trước đó.

Để tính tổng chiều cao của vùng tacần giá trị max y và min y của các cạnh gọi là miền xuất hiện của cạnh. Ta thực hiện chia nhỏ miền xuất hiện của cạnh thành các miền con và tính số lượng cạnh đang phủ lên trên miên này. Số lượng cạnh > 0 tức là tồn tại cạnh nào đó đang chứa miền con. Sau đó ta tính tổng độ dài các miền con được phủ.



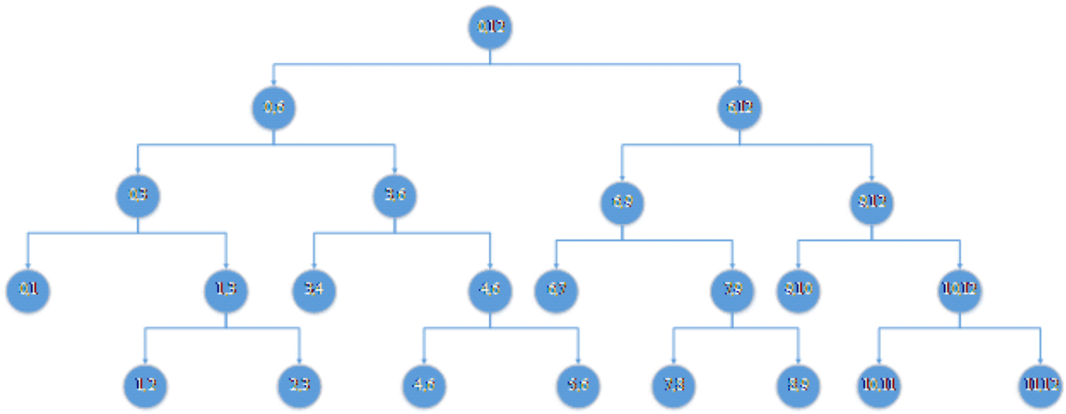
Hình 4. Tính max y, min y.

Việc chia các miền con được thực hiện bằng cách chia đôi những miền có chứa cạnh đến khi nào miền đó nằm hoàn toàn trong cạnh.



Hình 5. Chia miền con.

Quá trình chia miền có thể được biểu diễn bằng cây sau:



Hình 6. Cây phân đoạn.

Thuật toán

Dữ liệu vào: danh sách các cạnh được lưu trong mảng edge *canh; và số cạnh ở biến n. Mỗi cạnh được lưu trong cấu trúc như sau:

```
struct edge
{
   int x, y0, y1, d;
};
```

Trong đó: x là vị trí của cạnh trên trục Ox y0 là vị trí đầu của cạnh y1 là vị trí cuối của cạnh d là loại cạnh (1 là cạnh mở đầu, -1 là cạnh kết thúc)

Các bước của thuật toán

Bước 1: Chuẩn bị

1.1 Sắp xếp các cạnh theo thứ tự tăng dần của x

1.2 Tìm giá trị lớn nhất và nhỏ nhất của y, là maxy và miny

1.3 Dựa vào maxy và miny để cấp phát bộ nhớ cho v và sum

1.4 last = 0, dientich = 0, i = 0;

last lưu lại vị trí x cửa cạnh trước, dientich lưu lại tổng diện tích đã quét được, i biến đếm số cạnh.

Bước 2: nếu i == n thì nhảy đến bước 7

2.1 dientich += sum[0] * (canh[i].x - las); //thêm phần diện tích vừa quét vào ar

2.2 last = canh[i].x; // thay đổi cạnh củ

2.3 no = 0, no0 = miny, no1 = maxy;

2.4 y0 = canh[i].y0, y1 = canh[i].y1, d = canh[i].d

Bước 3: Nếu đoạn (no0, no1) nằm gọn trong đoạn (y0, y1)

Nếu đúng thì thay đổi số cạnh phủ miền (no0, no1)

v[no] = v[no] + d;

//nếu là cạnh mở đầu thì tăng lên 1

//nếu là cạnh kết thúc thì giảm 1

nhảy tới bươc 5

Nếu sai làm bước 4

Bước 4:

4.1 m = (no0 + no1) / 2; // chia đôi miền no0, no1

4.2 nếu đoạn (no0, m) và (y0, y1) giao nhau thì gọi đệ quy lại bước 3 với no' = no * 2 +1 , no0' = no0 và no1'= m

4.3 nếu đoạn (m, no1) và (y0, y1) giao nhau thì gọi đệ quy lại bước 3 với no' = no * 2 + 2 , no0' = m và no1' = no1

Bước 5: Cập nhật lại tổng

Nếu miền (no0, no1) có cạnh phủ (v[no] > 0)

Tổng độ dài cạnh trong miền (no0, no1) là độ dài đoạn (no0, no1) sum[no] = no1 - no0;

5.2 Ngược lại

Tổng độ dài cạnh trong miền (no0, no1) bằng tổng độ dài cạnh trong 2 miền con (no0, m) và (m, no1)

Bước 6: Tăng i lên 1 và quay lại bước 2 //nhảy đến cạnh kế tiếp

Bước 7: Kết thúc đưa ra dientich

Mã Chương trình

```
#include <iostream>
#include <fstream>
#include <algorithm>

#define max_n 30000

using namespace std;

struct edge
{
   int x, y0, y1, d;
};
struct sosanh
{
   bool operator() (edge x, edge y)
   {
      if (x.x == y.x)
         return ((x.d == -1) && (y.d == 1));
      return x.x < y.x;
   }
} myss;

edge *es;
int n, maxy = 0, miny = max_n;
long *v, *sum;

void doc()
```

```
{
   fstream f;
   int i, x0, x1, y0, y1;
   long c;
   f.open("mars.in", ios::in);
   f >> n;
   n *= 2;
   es = new edge[n];
   for (i = 0; i < n; i++)
   {
      f >> x0 >> y0 >> x1 >> y1;
      if (y0 < miny) miny = y0; //gia tri nho nhat cua y
      if (y1 > maxy) maxy = y1; //gia tri lon nhat cua y
      es[i].x = x0;
      es[i].y0 = y0;
      es[i].y1 = y1;
      es[i].d = 1;
      i++;
      es[i].x = x1;
      es[i].y0 = y0;
      es[i].y1 = y1;
      es[i].d = -1;
   }
   f.close();
   sort(es, es + n, myss);
   c = log(log(maxy - miny + 1) / log(2)) / log(2);
   c = long(powl(2, log(maxy - miny + 1) / log(2) + c + 3)); //tim so phan
       tu de khai bao cay can bang dua theo miny maxy
   v = new long[c];
   sum = new long[c];
   memset(v, 0, c); //set gia tri 0
   memset(sum, 0, c);
}
void correct(long long no, int no0, int no1)
{
   if (v[no] > 0) sum[no] = no1 - no0; //cong doan no0 no1 vao sum[no]
   else
   {
      sum[no] = 0;
      if (no0 < no1)
         sum[no] = sum[no * 2 + 1] + sum[no * 2 + 2]; //cap nhat lai nut phia
             tren
   }
}
void modify(long long no, int y0, int y1, int no0, int no1, int det)
{
   int m;
   if ((y0 <= no0) && (no1 <= y1))
      v[no] += det; // cay nhi phan can bang, cho biet so luong canh chong
          len mien gia tri tu no0 den no1, neu mien no0 no1 nam gon tron y0
          y1 thi cong 1 neu la canh dau hinh chu nhat -1 neu canh cuoi
   else
   {
      m = (no0 + no1) / 2; //be doi doan no0 no1
      if (y0 < m) //neu 1 phan cua doan y0 y1 co chua trong nua ben trai
         modify(no * 2 + 1, y0, y1, no0, m, det);
      if (m < y1) //neu 1 phan cua doan y0 y1 co chua trong nua ben phai
         modify(no * 2 + 2, y0, y1, m, no1, det);
```

```
    }
    correct(no, no0, no1); //cap nhat lai tong
}
int main()
{
    long long las = 0, ar = 0;
    fstream f;
    f.open("mars.out", ios::out);
    doc();
    for (int i = 0; i < n; i++)
    {
        ar += sum[0] * (es[i].x - las); //cong them dien tich mien vua quet
            duoc vao ar
        las = es[i].x; //luu lai vi tri x o canh truoc
        modify(0, es[i].y0, es[i].y1, miny, maxy, es[i].d);
    }
    f << ar;
    f.close();
    return 0;
}
```

2.2 Test đẹp và sự kế thừa

Bộ test của bài toán này bạn có thể download tại [1], chọn test Data của bài Mars Maps.

Test đẹp ở đây, tác giả muốn nói đó là có một bộ test với dữ liệu tạo ra biểu tượng của kỳ thi. Ngoài ra dữ liệu phân bố từ thấp đến cao rất đẹp. Tuy nhiên sau này nếu nghiên cứu nhiều về lớp bài toán này, các bạn sẽ nhận thấy bột test này chưa đạt nếu dữ liệu chuẩn hơn thì phương án gải của tác giả trên không thể accept được dẫn đến sau này một kỹ thuật mới "Lazy propagation" ra đời khác phục được nhược điểm trên.

Kỳ thi Olympic 2013 tại Đà nẵng, một bài toán có lẻ lấy ý tưởng từ bài toán này được đưa vào trong kỳ thi đó là bài toán "Trồng rau". Bộ test của bài toán này được đính kèm với bài đăng này, các bạn download về và thử.

Tài liệu

[1] Marcin Kubica, The 7th Baltic Olympiad in Informatics BOI 2001 Sopot, Poland. Tasks and Solutions

[2] Vũ Phúc Hoàng, Tự code, tự chấm, tự sướng. http://vnoi.info/wiki/algo/skill/viet-trinh-cham

[1] http://oi.edu.pl/boi2001/index0055.html?id=11