USACO JAN07 Problem 'schul' Analysis

by Bruce Merry / Richard Peng

Suppose that for a particular value of D, using the teacher's choices gives a ratio of M, and Bessie wants to improve this ratio. If she things of every test as being M plus or minus some number of marks (not percent), then she will want to drop the tests with the minimum number of marks over M, and keep those with the most marks over M. If it happens that the teacher's choice is different, then she can switch some of the tests to get more than 0 over M and thus improve her score.

A first attempt would be to verify this independently for each D. That is, for each D, determine the teacher-assigned mark, and look for the worst test kept in by the teacher and the best test dropped by the teacher ("better" meaning more marks above M). If substituting one for the other improves Bessie's score, then save this value of D. This gives an O(N^2) solution.

Quite a few contestants tried to insert heuristics into their code to improve runtime. Most of these relied on some assumption about the 'density' of the solution points and the locations where the min/max occurs. It was rather difficult to construct a data which fails both heuristics based on upper and lower bound, so two contestants still manged to get by with heuristic solutions.

There are several ways to go from here to get down to a O(nlogn) solution. First notice that the problem essentially reduces to finding the maximum/minimum values of the linear combination of a set of two variables. It suffices to consider the minimum case since the maximum case is a mirror equivalent. This can be thought as of either the x-intercepts of points on a plane or as the position where a certain line intersect a vertical line with a fixed x value. In both methods of visualization, it can be noticed that if a point is concave in relation to its two immediate neighbors, it can be deleted.

Both methods immediately leads to a O(nlogn) time solution using balanced binary trees. This can be explained in professor Brian Dean's animated notes found at: http://www.cs.clemson.edu/~bcdean/cowschool.swf This is also the only type of O(nlogn) solution implemented during the contest. Sorry to all the Pascal programmers as STL set gives C++ users a huge advantage should this solution be implemented. Nima Ahmadi Pour from Iran was the only person to implement one correctly (although Adrian Kuegel did come close). Nima's code is here:

```
#include <cstdlib>
#include <cstdio>
#include <algorithm>
#include <set>
#include <utility>
using namespace std;
double eps=1e-8;
FILE *fin=fopen("schul.in","r");
FILE *fout=fopen("schul.out","w");
struct test{
       int a,b;
```

```
	test(){}
	test(int t1,int t2):a(t1),b(t2){}
};
bool operator<(const test& t1,const test& t2){
	return t1.a*t2.b<t2.a*t1.b;
}
struct line{
	int a,b;
	line(){}
	line(int t1,int t2):a(t1),b(t2){}
};
bool operator<(const line& l1,const line& l2){
	return l1.b<l2.b;
}
set<line> s;
line last_line;
double last_x;
void reset(){
	s.clear();
}
double xinter(const line& l1,const line& l2){
	return double(l1.a-l2.a)/double(l2.b-l1.b);
}
double yinter(const line& l1,double x){
	return l1.b*x+l1.a;
}
double best(double x){
	double ans=-1e12;
	set<line>::iterator it=s.find(last_line);
	while (it!=s.end() && yinter(*it,x)>ans){
		ans=yinter(*it,x);
		++it;
	}
	--it;
	last_x=x;
	last_line=*it;
	return ans;
}
void insert(const line& l){
	while (true){
		set<line>::iterator it1=s.lower_bound(l);
		if (it1==s.end())
			break;
		set<line>::iterator it2=it1;++it2;
		if (it2==s.end())
			break;
		if (it1->b==l.b){
			if (it1->a<=l.a){
				s.erase(it1);
				continue;
			}
			return;
		}
		double x=xinter(*it1,*it2);
		if (yinter(l,x)<yinter(*it1,x)-eps)
			break;
		s.erase(it1);
```

```
	}
	while (true){
		set<line>::iterator it1=s.upper_bound(l);
		if (it1==s.begin())
			break;
		--it1;
		if (it1==s.begin())
			break;
		set<line>::iterator it2=it1;--it2;
		if (it1->b==l.b){
			if (it1->a<=l.a){
				s.erase(it1);
				continue;
			}
			return;
		}
		double x=xinter(*it1,*it2);
		if (yinter(l,x)<yinter(*it1,x)-eps)
			break;
		s.erase(it1);
	}
	set<line>::iterator it1=s.lower_bound(l);
	if (it1!=s.end() && it1!=s.begin()){
		set<line>::iterator it2=it1;--it2;
		double x=xinter(*it1,*it2);
		if (yinter(l,x)<=yinter(*it1,x)+eps)
			return;
	}
	s.insert(l);
	if (s.size()==1){
		last_line=l;
		last_x=-1e12;
	}
	else if (s.find(last_line)==s.end())
		last_line=l;
	else if (yinter(last_line,last_x)<yinter(l,last_x))
		last_line=l;
}
int n;
test t[50000];
double x[50000],min_one[50000],max_one[50000];
int main(){
	fscanf(fin,"%d",&n);
	for (int i=0;i<n;++i)
		fscanf(fin,"%d%d",&t[i].a,&t[i].b);
	sort(t,t+n);
	{
		int suma=0,sumb=0;
		for (int i=0;i<n;++i){
			suma+=t[n-i-1].a;
			sumb+=t[n-i-1].b;
			x[n-i-1]=double(suma)/double(sumb);
		}
	}
	{
		reset();
		for (int i=1;i<n;++i){
```

```
                        insert(line(t[i-1].a,-t[i-1].b));
                        max_one[i]=best(x[i]);
                }
        }
        {
                reset();
                for (int i=n-1;i>=1;--i){
                        insert(line(-t[i].a,-t[i].b));
                        min_one[i]=-best(-x[i]);
                }
        }
        int cou=0;
        for (int i=1;i<n;++i)
                if (min_one[i]<max_one[i]-eps)
                        ++cou;
        fprintf(fout,"%d\n",cou);
        for (int i=1;i<n;++i)
                if (min_one[i]<max_one[i]-eps)
                        fprintf(fout,"%d\n",i);
        return 0;
}
```

There exist a much simpler alternatives. One way is to divide the list in half, construct the hull for the lower half and query the score of every point from the higher half on the lower part in linear time. This would handle all the queries across the division point, which means we could recurse on each half to get to a O(nlogn) algorithm.

The other, much more elegant way by Bruce Merry is to consider what happens when we increment D and thus have a new point in our candidate set. Everything to the right (i.e. with higher T) can be discarded from the bag, because M will be (and remain) higher than the current test, and any tests to the right will be both of lower percentage and greater weight than the current test. Furthermore, if three elements in the bag are convex, then the middle one can never be optimal and it be discarded from the bag. This becomes a typical convex hull maintenance problem and can be done in O(N) time after the sorting. Bruce's code is here:

```
#include <fstream>
#include <algorithm>
#include <complex>
#include <vector>
#include <iterator>
#include <numeric>

using namespace std;

#define MAXN 50000

typedef long long ll;
typedef complex<ll> point;

static ll cross(const point &a, const point &b) { return imag(conj(a) *
b); }
static ll area2(const point &a, const point &b, const point &c)
{ return cross(b - a, c - a); }
```

```
struct compare_slope
{
    bool operator()(const point &a, const point &b)
    { return area2(point(0, 0), a, b) > 0; }
};

int main()
{
    static point points[MAXN];
    static point worst_in[MAXN];
    static point hull[MAXN];
    point total, slope;
    int N, H;
    vector<int> ans;
    ifstream in("schul.in");
    ofstream out("schul.out");

    in >> N;
    for (int i = 0; i < N; i++)
    {
        int p, t;
        in >> p >> t;
        points[i] = point(t, p);
    }
    sort(points, points + N, compare_slope());
    total = accumulate(points, points + N, point(0, 0));

    H = 0;
    slope = point(0, 0);
    for (int i = N - 1; i > 0; i--)
    {
        while (H && imag(hull[H - 1]) <= imag(points[i]))
            H--;
        while (H >= 2 && area2(hull[H - 2], hull[H - 1], points[i])
>= 0)
            H--;
        hull[H++] = points[i];
        slope += points[i];
        while (H >= 2 && cross(slope, hull[H - 2] - hull[H - 1])
<= 0)
            H--;
        worst_in[i - 1] = hull[H - 1];
    }

    H = 0;
    slope = total;
    for (int i = 0; i < N - 1; i++)
    {
        while (H && real(hull[H - 1]) >= real(points[i]))
            H--;
        while (H >= 2 && area2(hull[H - 2], hull[H - 1], points[i])
>= 0)
            H--;
        hull[H++] = points[i];
        slope -= points[i];
        while (H >= 2 && cross(slope, hull[H - 2] - hull[H - 1])
```

```
>= 0)
            H--;
        point best_out = hull[H - 1];
        if (cross(slope, worst_in[i] - best_out) < 0)
            ans.push_back(i + 1);
    }

    out << ans.size() << "\n";
    copy(ans.begin(), ans.end(), ostream_iterator<int>(out, "\n"));
    return 0;
}
```