Convex hull trick/acquire.cpp

From PEGWiki

```
/*
ID: brian_bi21
PROG: acquire
LANG: C++
*/

/*
6th line from the end
initially : if (i<N)
now       : if (i < N-1)
by : pktiw
*/

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope
vector<long long> B; //Holds the y-intercepts of the lines in the envelope
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
        /*
        intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
        intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
        set the former greater than the latter, and cross-multiply to
        eliminate division
        */
        return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
        //First, let's add it to the end
        M.push_back(m);
        B.push_back(b);
        //If the penultimate is now made irrelevant between the antepenultimate
        //and the ultimate, remove it. Repeat as many times as necessary
        while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
        {
                M.erase(M.end()-2);
                B.erase(B.end()-2);
        }
}
//Returns the minimum y-coordinate of any intersection between a given vertical
//line and the lower envelope
long long query(long long x)
{
        //If we removed what was the best line for the previous query, then the
        //newly inserted line is now the best for that query
        if (pointer>=M.size())
                pointer=M.size()-1;
        //Any better line must be to the right, since query values are
        //non-decreasing
        while (pointer<M.size()-1&&
          M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
                pointer++;
        return M[pointer]*x+B[pointer];
}
int main()
{
        int M,N,i;
        pair<int,int> a[50000];
        pair<int,int> rect[50000];
        freopen("acquire.in","r",stdin);
```

```
        freopen("acquire.out","w",stdout);
        scanf("%d",&M);
        for (i=0; i<M; i++)
                scanf("%d %d",&a[i].first,&a[i].second);
        //Sort first by height and then by width (arbitrary labels)
        sort(a,a+M);
        for (i=0,N=0; i<M; i++)
        {
                /*
                When we add a higher rectangle, any rectangles that are also
                equally thin or thinner become irrelevant, as they are
                completely contained within the higher one; remove as many
                as necessary
                */
                while (N>0&&rect[N-1].second<=a[i].second)
                        N--;
                rect[N++]=a[i]; //add the new rectangle
        }
        long long cost;
        add(rect[0].second,0);
        //initially, the best line could be any of the lines in the envelope,
        //that is, any line with index 0 or greater, so set pointer=0
        pointer=0;
        for (i=0; i<N; i++) //discussed in article
        {
                cost=query(rect[i].first);
                if (i < N-1)
                        add(rect[i+1].second,cost);
        }
        printf("%lld\n",cost);
        return 0;
}
```