
Convex Hull Trick

Convex Hull Trick is a data structure that can do two type of operation:

1. **Add Line**: Add a line to the set S. The line is of form $MX + B$, where M is gradient and B is y intersection.
2. **Query** X: Minimum value we get when lines in S are evaluated at position X.

Note: No line will be vertical.

Reference: wcipeg

Theory

If we draw all the lines on a paper, and then for each X highlight the point (X, Y), where Y is the lowest value we get if we evaluate all lines in set S with X, then we will get a upper-convex hull.

The upper-convex hull is all we need to answer our query. Therefore, if we can get the coordinates of the segments of the convex-hull, then we can easily evaluate for any value X using binary search.

There are two ways we can implement this DS: Offline and Online. The offline version is much simpler than online.

Offline Convex Hull Trick

If we are given all the lines and query beforehand, then we can pre-process the updates and queries easily.

The upper-covex hull will have segments from left to right in decreasing gradient. So, we will first sort all the lines in decreasing value of M. In case of tie, we will sort with increasing value of B.

Next, we will use a stack to maintain the subset of lines that form the upper-convex hull. The first line will always be in the stack. It will never be popped out.

Then for each line, either it will be pushed in the stack of it will be ignored. If the current line has same gradient as the top of stack, then ignore it, else push it into the stack. This new addition will make some lines in the convex hull invalid (that is they no longer take part in the hull). Find those invalid lines and remove them.

How do we find the invalid lines? Take the top 3 lines of the stack, l_1, l_2 and l_3. l_2 will become invalid if intersection of l_1 and l_2 is on right of intersection of l_1 and l_3.

Code

Used the following as reference: wcipeg

```
/* Instructions
1. Sort lines based on decreasing M and in case of tie, increasing B.
2. Sort query points according to increasing X.
3. Clear the class, add all lines and then query.
*/

class ConvexHullTrick {
    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 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]);
    }

    public:

    void clear() {
        pointer = 0;
        M.clear();
        B.clear();
    }

    //Adds a new line (with lowest slope) to the structure
    void add(long long m,long long b){
        if ( M.size() > 0 && M.back() == m ) return; ///Same Gradient. Don't add.

        //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){
        //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];
    }
};
```

Application

Minimum Dot Product

Given a set of points S and query points Q. For each point in Q, what is the minimum value we can get if we find its dot product with any point from S?

Basically, for any point (x, y) from Q, we have to find the minimum value of $A_i x + B_i y$, where (A_i, B_i) is a point from S.

Note that minimizing $A_i x + B_i y$ is same as minimizing $A_i \frac{x}{y} + B_i$, since y is a constant in every $A_i x + B_i y$.

Now, if we consider all points in S as lines having A_i gradient and B_i y-intercept, for each query point (x, y), we simply need to evaluate all the lines at $\frac{x}{y}$ and find the minimum value. However we will need to multiply this value with y to get desired dot product.

This can be solved using Convex Hull Trick. Note that we don't actually have to solve the problem for $X = \frac{x}{y}$ (doing so will introduce real numbers). When evaluating the line at `query()` function, simply evaluate it as $Mx + By$. This will ensure integer calculation and we won't even need to multiply y later.

Problem: CF Biathlon 2.0