GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: TRUCO DE LA ENVOLTURA CONVEXA (*CONVEX HULL TRICK*)

1. Introducción

Supongamos que un largo conjunto de funciones lineales de la forma $y = m_i x + b_i$ es dado junto con un largo número de pregunta. Cada pregunta consiste de un valor x y pregunta por el mínimo valor que puede ser obtenido si seleccionamos una de las funciones lineales y se evalúa en x. Por ejemplo, supongamos que nuestras funciones son $y = 4$, $y = \frac{4}{3} + \frac{3}{2}x$, $y = 12x - 3$, $y = 3 - \frac{1}{2}$ y recibimos la pregunta $x = 1$. Necesitamos identificar cual de esas funciones tiene el menor valor de y para $x = 1$, o cual es ese valor. (Es la función $y = \frac{4}{3} + \frac{3}{2}x$ tiene un valor de 2). Ver la imagen de abajo.

El truco de la envoltura convexa (*convex hull trick*) es una técnica (posiblemente mejor clasificado como una estructura de datos) usada para determinar eficientemente, después de un procesamiento, cuales miembros de un conjunto de funciones lineales de una variable alcanzan un valor extremo para un valor dado de la variable independiente.

2. Conocimientos previos

2.1. Producto punto

En matemáticas, el producto escalar, también conocido como producto interno o producto punto, es una operación algebraica que toma dos vectores y retorna un escalar, y que satisface ciertas condiciones. El producto escalar se define como la suma de los productos componente por componente de los dos vectores.

2.2. Envoltura Convexa (*Convex Hull*)

En matemáticas se define la envolvente convexa, envoltura convexa o cápsula convexa de un conjunto de puntos X de dimensión n como la intersección de todos los conjuntos convexos que contienen a X

En el caso particular de puntos en un plano, si no todos los puntos están alineados, entonces su

envolvente convexa corresponde a un polígono convexo cuyos vértices son algunos de los puntos del conjunto inicial de puntos.

Una forma intuitiva de ver la envolvente convexa de un conjunto de puntos en el plano, es imaginar una banda elástica estirada que los encierra a todos. Cuando se libere la banda elástica tomará la forma de la envolvente convexa.

2.3. Vector normal

En geometría un vector normal a una cantidad geometrica (línea, curva, superficie, etc) es un vector de un espacio de producto escalar que contiene tanto la entidad geométrica como al vector normal, que tiene la propiedad de ser ortogonal a todos los vectores tangentes a la entidad geométrica

3. Desarrollo

Una primera idea de solucíon sería para cada pregunta Q, simplemente evaluar cada una de las funciones del conjunto y determinar la que tenga menor valor para el valor x dado. Si M líneas son dadas junto con Q preguntas, la complejidad de esta solución es O(MQ). El truco que se presenta a continuación permite bajar la complejidad a O($(Q + M) \log M$).

Considere la figura de arriba. Note que la línea $y = 4$ nunca será mínima, independientemente del valor de x. Del resto de las tres líneas, cada una será la mínima en un solo continuo intervalo (posiblemente teniendo más o menos infinito por los límites). Podemos ver que la línea discontinua en verde es la mejor para todos los valores de x menores que la intersección con la verde continua más oscura, esta es la mejor entre esa intersección y su intersección con la verde continua más clara, y esta última es la mejor para valores de x más grandes. Note también que a medida que x incrementa, la pendiente de la línea mínima decrece:$\frac{2}{3}, -\frac{1}{2}, -3$. No es difícil ver que esto siempre ocurre.

Entonces, si eliminamos las líneas irrelevantes, tales como $y = 4$ en este ejemplo (líneas que nunca darán la mínima y-coordenada, independientemente de la pregunta) y ordenamos el resto de las líneas por sus pendientes, obtenemos una colección de N intervalos (donde N es el número de líneas restantes), cada uno perteneciente a una línea en el domino donde es la mínima. Si determinamos los puntos finales de esos intervalos, utilizando búsqueda binaria se responden cada una de las preguntas.

Como ya hemos visto, si el conjunto de relevantes líneas ha sido determinado y ordenado, es muy fácil responder las preguntas en O($\log N$) de complejidad usando búsqueda binaria. De esta manera si nosotros podemos añadir una línea cada vez a nuestra estructura de datos, recalculando esta información rápidamente con cada adición, tenemos un factible algoritmo: iniciar no con todas las líneas (una o dos dependiendo de los detalles de la implementación) y añadir líneas una por una hasta que todas las líneas hayan sido añadidas y nuestra estructura de datos está completa

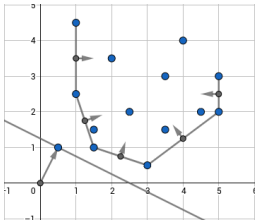
Supongamos que hemos sido capaz de procesar todas las líneas antes de necesitar la respuesta a cualquier pregunta. Entonces nosotros podemos ordenarlas descendentemente por pendiente

de antemano, y luego añadirlas una por una. Cuando se añade una nueva línea, algunas líneas pueden ser eliminadas porque ya no son más relevantes. Si nos imaginamos las líneas en una pila (contiene al elemento más reciente insertado en el tope de la pila), como añadimos cada nueva línea, consideramos si la línea en el tope de la pila ya no es relevante; si todavía lo es, insertamos nuestra nueva línea. Sino, sacamos de la pila la línea del tope y repetimos este procedimiento hasta que la línea del tope sea relevante o quede solo una línea en la pila

Cómo podemos determinar si la línea debe ser sacada de la pila?

Supongamos que l_1, l_2 y l_3 son la segunda línea del tope, la línea del tope y la línea a añadir respectivamente. Entonces, l_2 se convierte en irrelevante si y solo si el punto de intersección entre l_1 y l_3 está más a la izquierda que el de la intersección entre l_1 y l_2. Esto tiene sentido porque significa que el intervalo en el cual l_3 abarca es menor que el que abarcaba previamente l_2. Supongamos para mayor simplicidad que no hay tres líneas concurrente.

La idea de este enfoque es mantener una envoltura convexo inferior de funciones lineales. En realidad sería un poco más conveniente considerarlas no como funciones lineales, sino como puntos $(k; b)$ en el plano tal que tendremos que encontrar el punto que tiene el menor producto escalar con un punto dado $(x; 1)$, es decir, para este punto $kx + b$ se minimiza, que es lo mismo que el problema inicial. Tal mínimo necesariamente estará en una envoltura convexa más baja de estos puntos como se puede ver a continuación:



Uno tiene que mantener puntos en la envoltura convexa y vectores normales de los bordes de la envoltura convexa. Cuando usted tiene una $(x; 1)$ consulta tendrás que encontrar el vector normal más cercano a él en términos de ángulos entre ellos, entonces la función lineal óptima corresponderá a uno de sus puntos finales. Para ver eso, se debe notar que los puntos que tienen un producto escalar constante con $(x; 1)$ se encuentran en una línea que es ortogonal a $(x; 1)$, por lo que la función lineal óptima será aquella en la que tangente a la envoltura convexa que sea colineal con normal a $(x; 1)$ toca la envoltura. Este punto es aquel en el que las normales de las aristas situadas a la izquierda y a la derecha de él están encabezadas en diferentes lados de $(x; 1)$.

4. Implementación

4.1. C++

Para implementar este enfoque, uno debe comenzar con algunas funciones de utilidad geométrica, aquí sugerimos usar el tipo de número de complejo C++.

Aquí asumiremos que cuando se agregan funciones lineales, su K solo aumenta y queremos

encontrar valores mínimos. Mantendremos puntos en el vector $hull$ y vectores normales en el vector $vecs$. Cuando agregamos un nuevo punto, tenemos que mirar el ángulo formado entre el último borde en la envoltura convexa y el vector desde el último punto en la envoltura convexa hasta el nuevo punto. Este ángulo debe dirigirse en sentido antihorario, es decir, el producto DOT del último vector normal en la envoltura (dirigido dentro del casco) y el vector desde el último punto hasta el nuevo debe ser no negativo. Mientras esto no sea cierto, debemos borrar el último punto en la envoltura convexa junto con el borde correspondiente.

Ahora, para obtener el valor mínimo en algún momento, encontraremos el primer vector normal en la envoltura convexa que se dirige en sentido antihorario desde $(x; 1)$. El punto final izquierdo de dicho borde será la respuesta. Para verificar si el vector a no está dirigido en sentido antihorario de vector b, debemos verificar si su producto cruzado $[a, b]$ es positivo.

```
typedef int ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) { return (conj(a) * b).x(); }

ftype cross(point a, point b) { return (conj(a) * b).y(); }

vector<point> hull, vecs;

void add_line(ftype k, ftype b) {
   point nw = {k, b};
   while(!vecs.empty() && dot(vecs.back(), nw - hull.back()) < 0) {
      hull.pop_back(); vecs.pop_back();
   }
   if(!hull.empty()) { vecs.push_back(1i * (nw - hull.back())); }
   hull.push_back(nw);
}

int get(ftype x) {
   point query = {x, 1};
   auto it = lower_bound(vecs.begin(),vecs.end(),query,[](point a,point b) {
      return cross(a, b) > 0;
   });
   return dot(query, hull[it - vecs.begin()]);
}
```

4.2. Java

```
private class ConvexHullTrick{
   int size, ptr;

   private class Line{
```

```
        long m, n;
        public Line(long _m, long _n){
            this.m =_m; this.n =_n;
        }
    }

    Line [] lineMin;

    public ConvexHullTrick(int n) {
        size=ptr=0;
        lineMin= new Line[n];
        for(int i=0;i<n;i++) lineMin[i]= new Line(0L,0L);
    }

    public long query(long x) {
        ptr = Math.min(ptr, size - 1);
        while(ptr+1 < size &&
            lineMin[ptr+1].m*x+lineMin[ptr+1].n<lineMin[ptr].m*x+lineMin[ptr].n)
             ++ptr;
        return lineMin[ptr].m * x + lineMin[ptr].n;
    }

    public void add(Line line) {
        while(size>=2 && bad(lineMin[size-2],lineMin[size-1],line))--size;
        lineMin[size++] = line;
    }

    private boolean bad(Line l1, Line l2, Line l3) {
        return (l3.n-l1.n)*(l1.m-l2.m)<(l2.n-l1.n)*(l1.m-l3.m);
    }
}
```

5. Aplicaciones

La variación de esta técnica es buscar la máxima función, no la mínima. Esto se resuelve realizando una sencilla modificación, en está solamente nos concentramos en la mínima, ya que una vez entendido es fácil extrapolar a la máxima.

Este enfoque es útil cuando las consultas de suma de funciones lineales son monótonas en términos de k o si trabajamos fuera de línea, es decir, podemos agregar primero todas las funciones lineales y responder las consultas después. Eso requeriría manejar consultas en línea. Sin embargo, cuando se trata de consultas en línea, las cosas se pondrán difíciles y habrá que usar algún tipo de estructura de datos establecida para implementar una envoltura convexa adecuado. Sin embargo, el enfoque en línea no se considerará en esta guía debido a su dificultad y porque el segundo enfoque (que es el árbol de Li Chao) permite resolver el problema de manera más simple. Vale la pena mencionar que todavía se puede usar este enfoque en línea sin complicaciones por la descomposición de la raíz cuadrada. Es decir, reconstruir la envoltura convexa desde cero cada

$\sqrt{n}$ nuevas lineas.

Esta técnica es aplicada para optimizar problemas de programación dinámica cuando la recurrencia tiene la forma $dp[i] = min_{j<i} dp[j] + b[j] \times a[i]$ y se cumple que $b[j] \geq b[j+1]$ y $a[i] \leq a[i+1]$, logrando una complejidad temporal O(N)

6. Complejidad

Claramente, el espacio requerido es O(M) necesitamos solo almacenar el conjunto ordenado de lineas, las cuales son definidas por dos números reales. El tiempo requerido para ordenar todas las líneas por su pendiente es O($N \log N$). Cuando iteramos a través de ellas, añadiéndolas a la envolvente una por una, podemos notar que todas las líneas son insertadas en la pila exactamente una vez y que cada línea puede ser sacada a lo sumo una vez. Por esta razón, el tiempo requerido sobre todo es O(N) para este paso. El costo de ordenar domina, por lo que el tiempo de construcción es O($N \log N$).

7. Ejercicios propuestos

A continuación una lista de ejercicios que se pueden resolver aplicando esta técnica:

- DMOJ - Land Acquisition
- SPOJ - ACQUIRE - Land Acquisition
- DMOJ (no UCLV)- APIO '10 P1 - Commando
- CodeForce - C. Kalila and Dimna in the Logging Industry