Dualidad Punto-Línea y Convex Hull Trick

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp Argentina 2020

Contenidos

“[...] geometry, however, is not concerned with the relation of the ideas involved in it to objects of experience, but only with the logical **connection of these ideas among themselves**.”

Albert Einstein, *Relativity: The special and general theory*, 1920

“The convex hull trick is a technique [...] used to determine efficiently [...]. **It has little to do with convex hull algorithms**.”

Una de las referencias en bibliografía, que no sabe dualidad.

Contenidos

Cápsula convexa

- La forma que adopta una banda elástica, si la hacemos rodear "postes" clavados en los puntos.
- Se puede calcular con Graham Scan, en un solo recorrido.
- Se puede calcular en dos partes: La **cápsula superior** y la **cápsula inferior**.
- Ambos algoritmos se basan en "tener un stack e ir sacando hasta que encaje".

Contenidos

Problemas de ejemplo

- Cantidad de rectas “sobre las que pega la lluvia”
- Ancho de un conjunto de puntos en una cierta dirección
- Máxima cantidad de rectas concurrentes
- Cantidad de vértices de la cápsula convexa de puntos en el plano
- Máxima cantidad de puntos alineados
- “A qué altura pega la lluvia para un cierto x”

Problemas de ejemplo

- Cantidad de rectas “sobre las que pega la lluvia”
- Ancho de un conjunto de puntos en una cierta dirección
- Máxima cantidad de rectas concurrentes
- Cantidad de vértices de la cápsula convexa de puntos en el plano
- Máxima cantidad de puntos alineados
- “A qué altura pega la lluvia para un cierto x”

¿Cuáles de estos se relacionan?

Parejas de problemas duales

- Máxima cantidad de puntos alineados
- Máxima cantidad de rectas concurrentes

- Cantidad de vértices de la cápsula convexa de puntos en el plano
- Cantidad de rectas “sobre las que pega la lluvia”

- “A qué altura pega la lluvia para un cierto x”
- Ancho de un conjunto de puntos en una cierta dirección

Parejas de problemas duales

- Máxima cantidad de puntos alineados
- Máxima cantidad de rectas concurrentes

- Cantidad de vértices de la cápsula convexa de puntos en el plano
- Cantidad de rectas “sobre las que pega la lluvia”

- “A qué altura pega la lluvia para un cierto x”
- Ancho de un conjunto de puntos en una cierta dirección

¿Les parecen igual de fáciles/difíciles los emparejados?

Un ejemplo más (para tomarle sabor a la dualidad)

- Dados dos **puntos** y N **rectas**, cuántas **rectas** hay que cruzar como mínimo, al mover **un punto** hasta la posición del otro. NOTA: Es el K del dia 2 del TC 2020.
- Dadas dos **rectas no verticales** y N **puntos**, cuántos **puntos** hay que cruzar como mínimo, al mover **una recta** hasta la posición de la otra.
  La recta se "mueve" continuamente, como si fuera una especie de barra rígida infinita, puede girar y trasladarse en su camino. **Pero está prohibido en cualquier momento posicionarla en vertical.**

Contenidos

Representación de rectas en computadora

- Idea: ¿Cómo representamos una recta?

Representación de rectas en computadora

- Idea: ¿Cómo representamos una recta?
- Dependiendo del contexto y de cómo la vamos a usar hay muchas maneras.
- Para "cálculos geométricos con rectas y figuras", en general representaciones con **vectores** es lo mejor.

Representación de rectas en computadora

- Idea: ¿Cómo representamos una recta?
- Dependiendo del contexto y de cómo la vamos a usar hay muchas maneras.
- Para “cálculos geométricos con rectas y figuras”, en general representaciones con **vectores** es lo mejor.
- Si la recta representa naturalmente una cierta *función lineal* que a veces evaluamos, la **forma explícita** $y = ax + b$ puede ser más útil.
- Notar que de ser así no tienen sentido las rectas verticales (no son funciones).
- Salvo cuando aclaremos lo contrario, “recta” para nosotros será en particular durante toda esta charla “recta no vertical”.

Rectas y puntos en struct

```
struct Recta
{
    int a,b; // Forma explicita:
             // contiene los puntos (x,y)
             // que cumplen y = ax + b
};

struct Punto
{
    int x,y;
};
```

Rectas y puntos en struct

```
struct Recta
{
    int a,b; // Forma explicita:
             // contiene los puntos (x,y)
             // que cumplen y = ax + b
};

struct Punto
{
    int x,y;
};
```

- ¿Qué diferencia hay entre ambas?

Rectas y puntos en struct

```
struct Recta
{
    int a,b; // Forma explicita:
             // contiene los puntos (x,y)
             // que cumplen y = ax + b
};

struct Punto
{
    int x,y;
};
```

- ¿Qué diferencia hay entre ambas?
- Los nombres. “Names Don’t Constitute Knowledge” R. Feynman
  https://www.youtube.com/watch?v=lFIYKmos3-s

La transformación

- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.

La transformación

- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.
- ¿Eso es todo? ¿Eso es dualidad punto recta?

La transformación

- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.
- ¿Eso es todo? ¿Eso es dualidad punto recta?
- Sí.

La transformación

- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.
- ¿Eso es todo? ¿Eso es dualidad punto recta?
- Sí.
- ¿De verdad?

La transformación

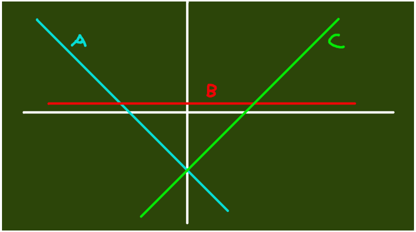
- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.
- ¿Eso es todo? ¿Eso es dualidad punto recta?
- Sí.
- ¿De verdad?
- Sí.

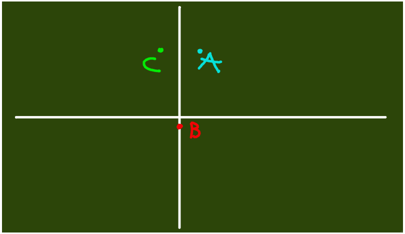
La transformación

- Transformar rectas en puntos: $y = ax + b \mapsto (-a, -b)$
- Transformar puntos en rectas: $(a, b) \mapsto y = -ax - b$
- En términos de los structs anteriores, siempre es $(f_1, f_2) \mapsto (-f_1, -f_2)$, más allá de los nombres de los *fields* f_1, f_2 en el struct que corresponda.
- ¿Eso es todo? ¿Eso es dualidad punto recta?
- Sí.
- ¿De verdad?
- Sí.
- Pero para que sea útil hay que entender la **relación** entre ellos.

Relaciones

- **No** transforma el plano como una rotación/traslación/reflexión: **puntos no van a puntos.**
- Transforma puntos y rectas, e **intercambia sus roles.**
- Por esta razón, una forma de ilustrar la transformación es mediante colores:





Aproximadamente:

- A) $f(x) = y = -x - 5 \leftrightarrow p = (1, 5)$
- B) $f(x) = y = 1 \leftrightarrow p = (0, -1)$
- C) $f(x) = y = x - 5 \leftrightarrow p = (-1, 5)$

Relaciones

- $y = f(x) = ax + b$, y $(a, b) = p$, la dualidad intercambia p y f
- Al dualizar dos veces volvemos al mismo punto / recta de partida

Relaciones

- $y = f(x) = ax + b$, y $(a, b) = p$, la dualidad intercambia p y f
- Al dualizar dos veces volvemos al mismo punto / recta de partida
- "rectas no verticales" = " funciones lineales" $\Leftrightarrow$ "puntos del plano"

Relaciones

- $y = f(x) = ax + b$, y $(a, b) = p$, la dualidad intercambia p y f
- Al dualizar dos veces volvemos al mismo punto / recta de partida
- "rectas no verticales" = " funciones lineales" $\Leftrightarrow$ "puntos del plano"
- "pendiente mayor" $\Leftrightarrow$ "coordenada x menor"

Relaciones

- $y = f(x) = ax + b$, y $(a, b) = p$, la dualidad intercambia p y f
- Al dualizar dos veces volvemos al mismo punto / recta de partida
- "rectas no verticales" = " funciones lineales" $\Leftrightarrow$ "puntos del plano"
- "pendiente mayor" $\Leftrightarrow$ "coordenada x menor"
- "rectas paralelas" $\Leftrightarrow$ "alineados en vertical"

Operador está arriba/abajo

- Si $p = (x_p, y_p)$, R es $y = ax + b$, decimos $p \uparrow R$ si $y_p \geq ax_p + b$
- Lo leemos " p está arriba de R "
- Análogamente $p \downarrow R$
- Observación: $p \in R \Leftrightarrow p \uparrow R \wedge p \downarrow R$

Relación esencial de la dualidad punto-línea

- $p \uparrow R \Leftrightarrow dual(R) \uparrow dual(p)$
- Corolario: $p \in R \Leftrightarrow dual(R) \in dual(p)$

Relación esencial de la dualidad punto-línea

- $p \uparrow R \Leftrightarrow dual(R) \uparrow dual(p)$
- Corolario: $p \in R \Leftrightarrow dual(R) \in dual(p)$
- Demostración:
  - $p \uparrow R$ es por definición $y_p \geq ax_p + b$
  - $dual(R) \uparrow dual(p)$ es por definición $-b \geq (-x_p)(-a) + (-y_p)$
  - Ambas condiciones son equivalentes
  - Nota: esta es la razón para poner los signos negativos en la transformación

Relaciones de incidencia

De lo anterior tenemos:

- "muchos puntos arriba de una recta" $\Leftrightarrow$
  "muchas rectas que pasan por debajo de un punto"
- "rectas concurrentes" $\Leftrightarrow$ "puntos alineados (pero no en vertical)"

Relaciones de extremos

- "y" (o sea $f(x)$ dado x) $\Leftrightarrow p \cdot (-x, -1)$

Relaciones de extremos

- "y" (o sea $f(x)$ dado x) $\Leftrightarrow p \cdot (-x, -1)$
- "Recta con mayor $f(x)$" $\Leftrightarrow$ "Punto que está más avanzado en la dirección apuntada por $\vec{v} = (-x, -1)$".

Relaciones de extremos

- "y" (o sea $f(x)$ dado x) $\Leftrightarrow p \cdot (-x, -1)$
- "Recta con mayor $f(x)$" $\Leftrightarrow$ "Punto que está más avanzado en la dirección apuntada por $\vec{v} = (-x, -1)$".
- **"Envolvente superior" $\Leftrightarrow$ "Cápsula inferior".**
- **"Envolvente inferior" $\Leftrightarrow$ "Cápsula superior".**

Relaciones de extremos

- "y" (o sea $f(x)$ dado x) $\Leftrightarrow p \cdot (-x, -1)$
- "Recta con mayor $f(x)$" $\Leftrightarrow$ "Punto que está más avanzado en la dirección apuntada por $\vec{v} = (-x, -1)$".
- **"Envolvente superior"** $\Leftrightarrow$ **"Cápsula inferior".**
- **"Envolvente inferior"** $\Leftrightarrow$ **"Cápsula superior".**
- Esta última es la que da el nombre a "convex hull trick"
- ¡Revisar los problemas anteriores para verificar los emparejamientos!

Contenidos

Qué es

- Es la técnica que usamos para calcular eficientemente,a partir de un x, el valor de $max_{i=1}^{n} f_i(x)$, si tenemos n funciones lineales f_i

Qué es

- Es la técnica que usamos para calcular eficientemente,a partir de un x, el valor de $max_{i=1}^{n} f_i(x)$, si tenemos n funciones lineales f_i
- Permite mantener la envolvente superior de rectas eficientemente.

Qué es

- Es la técnica que usamos para calcular eficientemente,a partir de un x, el valor de $max_{i=1}^{n} f_i(x)$, si tenemos n funciones lineales f_i
- Permite mantener la envolvente superior de rectas eficientemente.
- Por dualidad, **es equivalente tener la cápsula inferior de puntos**, que son los únicos puntos candidatos a ser los extremos en una dirección $\vec{v} = (-x, -1)$.

Convex Hull Trick offline

- Si las rectas van apareciendo **en orden de pendiente**, o si las podemos ordenar libremente, hacemos lo mismo que para construir cápsula inferior/superior si los puntos están ordenados por x
- Ante cada nueva recta, vamos “descartando hasta que encaje” del vector de rectas, y siempre tendremos las rectas de la envolvente **ordenadas** (binary search).

Convex Hull Trick online

- Si las rectas aparecen en forma dinámica y en cualquier orden, podemos tenerlas en un `set` **ordenadas por pendiente**, y usar `lower_bound` para ver dónde insertar la nueva (si es que hay que hacerlo).

Convex Hull Trick online

- Si las rectas aparecen en forma dinámica y en cualquier orden, podemos tenerlas en un `set` **ordenadas por pendiente**, y usar `lower_bound` para ver dónde insertar la nueva (si es que hay que hacerlo).
- Podemos usar la dualidad para razonar más fácil en términos de puntos (si así nos resulta más fácil), pues es análogo a mantener la cápsula inferior cuando aparecen nuevos puntos: allí los mantendríamos ordenados por coordenada x, y luego "borraríamos adyacentes a cada lado" hasta que la banda elástica quede "correctamente tensada".

Convex Hull Trick online

- Si las rectas aparecen en forma dinámica y en cualquier orden, podemos tenerlas en un `set` **ordenadas por pendiente**, y usar `lower_bound` para ver dónde insertar la nueva (si es que hay que hacerlo).
- Podemos usar la dualidad para razonar más fácil en términos de puntos (si así nos resulta más fácil), pues es análogo a mantener la cápsula inferior cuando aparecen nuevos puntos: allí los mantendríamos ordenados por coordenada x, y luego "borraríamos adyacentes a cada lado" hasta que la banda elástica quede "correctamente tensada".
- O más aún podemos programar directamente el código para mantener la cápsula inferior, y traducir (dualizar) los parámetros de las queries.

Convex Hull Trick online (implementación)

- La implementación es muy delicada:
  - Para las inserciones, queremos hacer lower-bound por (pendiente de recta / x del punto)
  - Para las queries, queremos hacer lower-bound por ("derivada" de $f(x)$ / derivada de $p \cdot (-x, -1)$)
  - Si bien el valor que damos es distinto, con ambos criterios el conjunto está bien ordenado
  - Hack para set de C++: Se puede tener un comparator que cambia su comportamiento según variables globales
- Recomendación: sentarse con tiempo y mucho cuidado a entender la idea e implementarla, y guardar la implementación para usar en competencias

Contenidos

Nos faltan puntos

- Tendríamos que poder dualizar las rectas verticales, pero ya usamos todos los puntos del plano.
- Solución: **Agregar** puntos al plano.
- ¿Dónde tendría que estar el punto dual de una recta vertical?

Nos faltan puntos (cont.)

- Por el punto $dual(R)$, pasan todas las rectas $dual(p)$ para los puntos $p \in R$.
- ¡Si R es vertical, esos puntos dualizan a rectas paralelas!
- Necesitamos agregar puntos en el infinito, donde las rectas paralelas se cortan.
- Por cada posible pendiente de rectas paralelas, habrá un punto en el infinito.
- La recta vertical $x = a$ dualiza al punto infinito donde convergen las rectas de pendiente $-a$.
- ¿Dónde se cortan las rectas verticales?

Nos faltan puntos (cont.)

- Se cortan en otro punto en el infinito, que no tiene todavía recta dual...
- La recta dual del “infinito vertical” es una “recta en el infinito”, que pasa exactamente por todos los puntos infinitos.

Plano proyectivo

- Lo anterior hace que toda recta y todo punto tenga dual, y mantiene la relación esencial de dualidad punto línea.
- El plano con los puntos y rectas que agregamos se llama plano proyectivo.
- Una forma equivalente de construirlo (pero con un cambio de coordenadas) sin separar como especial la recta “infinita” es pensar que tomamos:
  - Los subespacios de dimensión 1 en $\mathbb{R}^3$, como puntos
  - Los subespacios de dimensión 2 en $\mathbb{R}^3$, como rectas
  - Dualizar es tomar complemento ortogonal
- En general, estos conceptos no se usan en programación competitiva :)

Un ejemplo: Problema el K en el plano proyectivo

Los siguientes problemas son duales (al considerarlo en el plano proyectivo)

- Dados dos **puntos** y N **rectas**, cuántas **rectas** hay que cruzar como mínimo, al mover **un punto** hasta la posición del otro, pero se puede “dar la vuelta por el infinito”.
- Dadas dos **rectas** y N **puntos**, cuántos **puntos** hay que cruzar como mínimo, al mover **una recta** hasta la posición de la otra.

En este caso, la “circularidad” que complica el problema natural de rectas, se vuelve explícita al dualizar.

Contenidos

Bibliografía

- http://wcipeg.com/wiki/Convex_hull_trick
- https://en.wikipedia.org/wiki/Duality_(projective_geometry)
- https://en.wikipedia.org/wiki/Projective_plane#Plane_duality
- http://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect06-duality.pdf