Universidad Nacional Autónoma de México
Facultad de Ciencias

Geometría Computacional

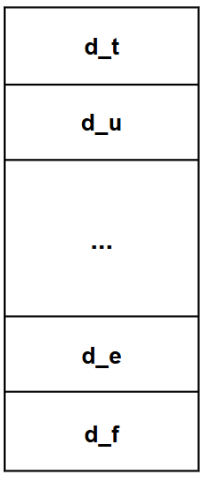
Semestre: 2025-2

Tarea 2

García Ponce José Camilo 319210536

Ejercicio 1

Si es posible calcular el cierre convexo en menor tiempo, para esto usaremos el algoritmo de Melkman (el mismo usado para el ejercicio 4 de la primera tarea), veamos cómo funciona, suponemos que los vértices del polígono están en posición general.
Como tenemos a nuestro polígono simple P entonces vamos a tener los vértices que lo conforman en una especie de lista circular (esto es una forma de representar al polígono pero puede ser cualquier otra) y además usaremos una deque (cola doblemente terminada) donde podemos agregar y eliminar elementos al inicio y final de la cola, para facilitar la explicación del algoritmo tendremos que d_t es el vértice en el tope de la deque, d_u será el segundo vértice en el tope de la de que (esta abajo de d_t), d_f será el vértice al final de la deque y d_e será el penúltimo vértice de la deque (esta arriba de d_f), notemos que los vértices dentro de la deque serán los vértices que representan el cierre convexo de los vértices procesados.



Entonces tomamos a los tres primeros vértices del polígono simple los cuales son $v1$, $v2$ y $v3$, ahora revisamos si $v3$ está a la derecha del segmento de recta de $v1$ a $p2$, si está a la derecha entonces metemos a $v1$ al tope de la deque y luego metemos a $v2$ en el tope, pero si $v3$ no esta a la derecha entonces metemos a $v2$ en el tope de la deque y luego metemos a $v1$ en el tope, después de esto vamos a meter a $v3$ al tope y fin de la deque.
Posteriormente, vamos a procesar los demás vértices del polígono y cuando lleguemos al primer vértice del polígono ya vamos a haber acabado (notemos ese vértice ya no lo procesamos, es decir cuando volvamos a encontrar a $v1$ terminamos el algoritmo).
Veamos cómo se va a procesar cada vértice, sea v el próximo vértice del polígono a procesar, lo primero que hacemos es revisar si v está dentro de la envolvente convexa que llevamos de momento, esto lo hacemos revisando si se cumple que el vértice d_e está a la izquierda del segmento de recta del vértice v al vértice d_f o que

el vértice v este a la izquierda del segmento de recta del vértice d_u al vértice d_t, si no se cumple nada de lo anterior entonces significa que v se encuentra dentro de la envolvente y terminamos de procesar a v, pero si se cumple al menos una entonces ahora vamos a revisar si se cumple que el vértice v está a la derecha del segmento de recta del vértice d_u al vértice d_t, si no se cumple entonces sacamos el elemento del tope de la deque y volvemos a revisar si se cumple, esto lo repetimos hasta que se cumpla que el vértice v está a la derecha del segmento de recta del vértice d_u al vértice d_t, de esta manera cuando se cumpla (puede ser desde la primera vez y entonces no sacamos nada de la deque) vamos a agregar el vértice v en el tope de la deque, después de eso vamos a revisar si se cumple que el vértice d_e está a la derecha del segmento de recta del vértice v al vértice d_f, si no se cumple entonces sacamos el elemento del final de la deque y volvemos a revisar si se cumple, esto lo repetimos hasta que se cumpla que vértice d_e está a la derecha del segmento de recta del vértice v al vértice d_f, de esta manera cuando se cumpla (puede ser desde la primera vez y entonces no sacamos nada de la deque) vamos a agregar el vértice v en el final de la deque, y con esto terminamos de procesar al vértice v (en esta parte volví a explicarlo de igual manera que en la tarea 1 ya que es asi como esta explicado en el documento que está en classroom sobre Melkman, pero también funciona usando la dirección de giro de v con el segmento de recta del vértice d_e al vértice d_f, comentario en la tarea anterior).

Procesamos todos los vértices del polígono hasta llegar al primer vértice de este.

Al final de procesar todos los vértices tendremos los vértices que forman el cierre convexo dentro de la deque y solo los sacamos.

Notemos que esto funciona ya que se van agregando vértices al cierre convexo poco a poco, por eso ignoramos a los vértices que ya se encuentran dentro y al momento de agregar nuevos vértices revisamos los últimos vértices agregados a la deque para sacar los vértices que no van a formar parte del cierre convexo, esto para que se cumpla que si viajamos por los vértices del cierre convexo en una dirección solo hacemos giros/vueltas a la derecha y en la otra dirección solo son giros/vueltas a la izquierda.

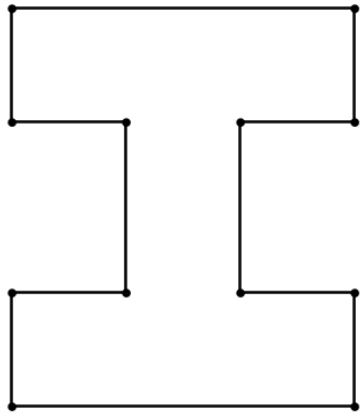
Además el algoritmo sirve ya que en el deque tenemos el cierre convexo de los vértices que ya hemos procesado, por lo tanto al procesar un nuevo vértice la deque se modifica y por lo tanto el cierre convexo también.

Ahora veamos la complejidad de todo, observemos que ver la dirección de un vértice con respecto de un segmento de recta lo podemos hacer en tiempo constante (como lo vimos en clase), también la operación de agregar o sacar del tope o final de la deque se puede hacer en tiempo constante y ver el segundo elemento del tope o penúltimo del final también se puede hacer en constante, de esta manera el proceso de los tres primeros vértices lo hacemos en $O(1)$, luego el proceso de los demás vértices nos tomara $O(n)$, ya que cada vértice lo metemos a

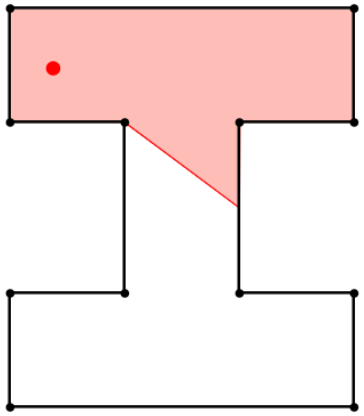
la deque a lo más dos veces y lo sacamos a lo más dos veces, por lo cual en total procesar todos los vértices nos toma $O(n)$ (cada punto entra y sale a lo más dos veces a la deque y esto nos genera la complejidad $O(n)$, ya los vértices ya sacados de la deque ya no van a poder regresar a la deque) y al final sacar todos los vértices de la deque para conocer el cierre convexo nos toma $O(n)$, de esta manera podemos ee todo nos toma $O(n)$ y se cumple que obtenemos el cierre convexo de un polígono en tiempo menor que $O(n\ log\ n)$.
Este es el mismo algoritmo usado para el ejercicio 4 de la primera tarea ya que en ambos ejercicios se busca el cierre convexo y además una poligonal simple cerrada es un polígono.

Ejercicio 2

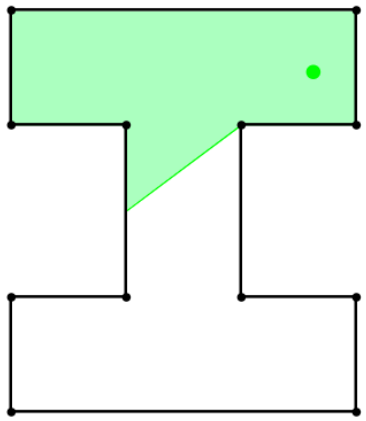
Esto no siempre pasa, veamos un contraejemplo.
Tengamos este polígono P.



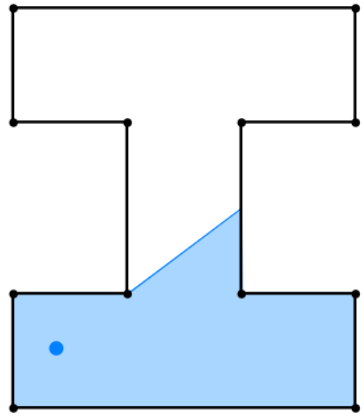
Y con este conjunto de cámaras (los puntitos).
La primera cámara será esta, en rojito está coloreada la parte que puede ver de P.



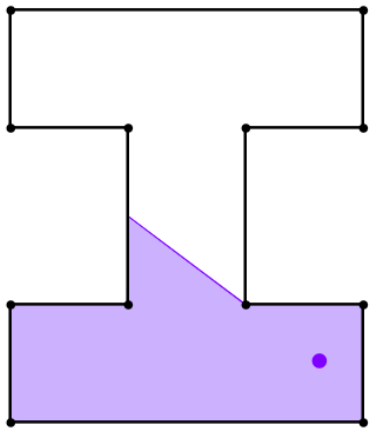
La segunda cámara será esta, en verdecito está coloreada la parte que puede ver de P.



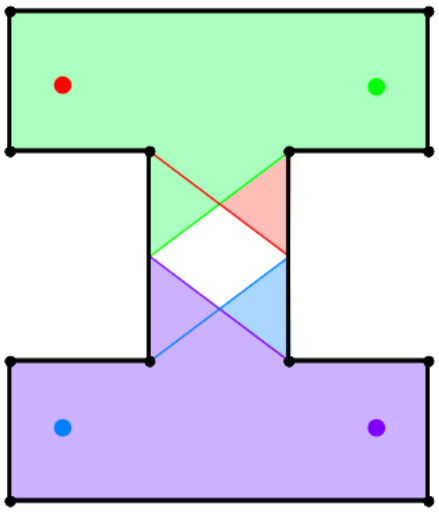
La tercera cámara será esta, en azulito está coloreada la parte que puede ver de P.



Y la cuarta cámara será esta, en moradito está coloreada la parte que puede ver de P.



Ahora si ponemos las cuatro cámaras en P, podemos observar que todo el borde P está siendo observado por este conjunto de cámaras, pero en el centro del polígono tenemos una región que no la puede observar ninguna cámara (la parte en blanco), por lo tanto este en un contraejemplo ya que tenemos un polígono simple y un conjunto de cámaras que observan todo el borde del polígono pero no observan todo el interior del polígono.



Esto se debe a que por la posición de las cámaras y nuestras aristas se generan partes del polígono simple las cuales no son visibles por las cámaras aunque todo

todo el borde del polígono si es visible por las cámaras, y esto surge ya que las cámaras que pusimos tienen diferentes partes del polígono que no pueden observar y la parte que ninguna cámara puede observar es una intersección de todas las partes que no pueden observar nuestras cámaras.

Ejercicio 3

Para este ejercicio usaremos el algoritmo (o los algoritmos) visto en la exposición del 12/05/25.
Tendremos un polígono P con n vértices y un punto q, para obtener el polígono de visión del punto x en el polígono P vamos a realizar los siguientes pasos.
Notemos que el número de revoluciones del polígono P con respecto al punto q es el número de revoluciones que el límite del polígono hace alrededor de q, si el número de revoluciones es una, entonces el polígono no tiene winding, para determinar el winding de un polígono podemos lanzar un rayo desde el punto q y luego recorrer las aristas del polígono para ver cuántas aristas cruzan al rayo en sentido contrarreloj y cuantas en sentido de las manecillas del reloj, y por último restar estas cantidades.
Lo primero es revisar si el polígono tiene winding o no, en caso de que si vamos a usar el primer algoritmo de la exposición (este algoritmo es algo complejo y difícil, así que vamos a dar una explicación no tan detallada).
Lo primero que vamos a hacer es trazar una línea horizontal a la izquierda del punto q la cual será L_l y una línea horizontal a la derecha del punto y será L_r, tendremos que el punto de intersección entre el polígono y L_l lo vamos a conocer como q_l (es la intersección más cercana con q), de manera análoga para L_r tendremos al punto q_r, usando estos puntos vamos a generar el polígono P_a que sera el polígono formado del recorrido contrarreloj iniciado en q_r y terminan en q_l y el polígono P_b que sera el polígono formado del recorrido en sentido de las manecillas del reloj iniciado en q_r y terminan en q_l, con estas cosas ya podemos empezar los pasos del algoritmo.
Para el algoritmo tendremos cuatro procesos, los cuales son $C(P_a, q_l, q_r, L_l)$, $C(P_b, q_r, q_l, L_r)$, $CC(P_b, q_l, q_r, L_l)$ y $CC(P_a, q_r, q_l, L_r)$, los cuatro casos son análogos (solo cambiar qué línea horizontal vamos a usar, qué polígono usamos y de dónde a dónde vamos a recorrer los puntos del polígono) y por lo tanto veamos cómo se hace $CC(P_a, q_r, q_l, L_r)$, el objetivo de estos procesos es obtener las aristas que tenemos que agregar a P_a y P_b, con el objetivo de que al quitar de P_a las partes que están debajo de las aristas agregadas del proceso $C(P_a, q_l, q_r, L_l)$ y $CC(P_a, q_r, q_l, L_r)$ tendremos que ahora el nuevo P_a siempre estará encima de L_l y L_r (ya no por debajo), de manera análoga para P_b tendremos que nos queda que siempre estará por debajo de L_l y L_r, y así al unir P_a y P_b, con las modificaciones, tendremos un polígono sin winding.
Para el proceso usaremos una pila.

Paso 1, calculamos los puntos de intersección entre L_r y el límite de P_a, los cuales los vamos a llamar como z_0, z_1, …, z_m, donde z_0 es q_r y z_m es q_l, y al conjunto total de puntos de intersección lo llamamos Z. Inicializamos una variable i en 1 y otra variable h en 0.
Paso 2, vamos a recorrer los puntos de intersección, iniciando en z_i que sera z_1 al inicio.
Paso 3, revisamos si z_i es un punto descendiente (un punto z de intersección entre L_r y P_a es descendiente si el siguiente vértice en sentido contrarreloj de z en P_a está por abajo de L_r) y no pertenece al segmento de recta del vértice q al vértice z_h, entonces vamos a recorrer los puntos de Z a partir de z_{i+1} hasta encontrar un punto z_k que esté en el segmento de recta del vértice q al vértice z_i, luego de encontrar el punto z_k vamos a revisar si z_k es un punto ascendiente (un punto z de intersección entre L_r y P_a es ascendiente si el siguiente vértice en sentido contrarreloj de z en P_a está por arriba de L_r), si es ascendiente entonces metemos el segmento de recta del vértice z_i al vértice z_k en la pila y actualizamos el valor de i como $k + 1$, pero en caso de que v_k sea descendiente entonces vamos a recorrer los puntos de Z a partir de z_{l+1} hasta encontrar un punto z_j que esté en el segmento de recta del vértice z_i al vértice z_k, luego metemos el segmento de recta del vértice z_i al vértice z_j en la pila y actualizamos el valor de i como $j + 1$. Después de realizar esto revisamos si todos los puntos de Z fueron recorridos, si todos fueron recorridos entonces pasamos al Paso 11, pero si no entonces actualizamos el valor de h a $i - 1$ y pasamos al Paso 3.
Paso 4, revisamos si z_i es un punto descendiente y pertenece al segmento de recta del vértice q al vértice z_h, entonces tendremos que el segmento de recta del vértice z_k al vértice z_r será el tope de la pila, para facilitar las cosas llamaremos al tope de la pila como g, lo que vamos a hacer es quitar el tope de la pila hasta que se cumple que z_i pertenece a g, luego vamos a recorrer los puntos de Z a partir de z_{i+1} hasta encontrar un punto z_j que esté en el segmento de recta del vértice z_k al vértice z_i, al encontrarlo vamos a quitar el tope de la pila e insertamos a la pila el segmento de recta del vértice z_k al vértice z_j. Después de realizar esto revisamos si todos los puntos de Z fueron recorridos, si todos fueron recorridos entonces pasamos al Paso 11, pero si no entonces actualizamos el valor de i a $j + 1$ y el de h a $i - 1$ y pasamos al Paso 3.
Paso 5, revisamos si z_i es un punto ascendiente y pertenece al segmento de recta del vértice q al vértice z_h, vamos a recorrer los puntos de Z a partir de z_{i+1} hasta

encontrar dos puntos consecutivos descendientes z_{j-1} y z_j que cumplan que ambos puntos estén en el segmento de recta del vértice q al vértice z_h, si los encontramos entonces actualizamos i al valor de j y pasamos al Paso 4, pero ni no los encontramos entonces vamos a particionar a P_a al agregar las aristas o segmentos de recta que tenemos en la pila y vaciamos la pila, luego vamos a calcular los puntos de intersecciones entre el segmento de línea del vértice q al vértice z_i y el límite del polígono formado del vértice z_i al vértice q_l, estas intersecciones serán w_0, w_1, …, w_p (tal que w_0 es z_i), los llamaremos W y pasamos al Paso 7.

Paso 6, revisamos si z_i es un punto ascendiente y no pertenece al segmento de recta del vértice q al vértice z_h, entonces calculamos los puntos de intersecciones entre el segmento de línea del vértice z_h al vértice z_i y el límite del polígono formado del vértice z_i al vértice q_l, estas intersecciones serán w_0, w_1, …, w_p (tal que w_0 es z_i), los llamaremos W y limpiamos la pila.

Paso 7, metemos el segmento de recta del vértice w_0 al vértice w_1 a la pila e inicializamos una variable k en 2.

Paso 8, revisamos si el vértice w_k no está en el segmento de recta del vértice w_0 al vértice w_{k-1}, si se cumple entonces vamos a recorrer los puntos de W a partir de w_{k+1} hasta encontrar un punto w_j que cumpla con que w_j no esté en el segmento de recta del vértice w_0 al vértice w_k, al encontrarlo vamos a meter al segmento de recta del vértice w_k al vértice w_j a la pila, actualizamos k a $j + 1$ y pasamos al Paso 10.

Paso 9, revisamos si el vértice w_k está en el segmento de recta del vértice w_0 al vértice w_{k-1}, si se cumple entonces sea el segmento de recta del vértice w_j al vértice w_t el tope de la pila, por facilidad lo llamaremos f, vamos a quitar el tope de la pila hasta que w_k este en f, después vamos a recorrer los puntos de W a partir de w_{k+1} hasta encontrar un punto w_r que esté en el segmento de recta del vértice w_j al vértice w_k, al encontrarlo sacamos el tope de la pila y metemos a la pila el segmento de recta del vértice w_j al vértice w_r.

Paso 10, si todos los puntos de W no han sido recorridos o revisados aun entonces pasamos al Paso 8.

Paso 11, particionamos a P_a al agregar las aristas o segmentos de recta que tenemos en la pila.

Luego de realizar los cuatro procedimientos y juntar los polígonos resultantes vamos a poder tener el polígono sin winding, esto funciona ya que en los procedimientos sobre P_a vamos revisando cuando se intersecta con L_r y L_l para poder agregar las

aristas necesarias para que al modificar P_a siempre se quede arriba de las líneas horizontales, de manera similar para P_b solo que quede abajo y así evitar que el polígono resultante tenga más de una revolución sobre el punto q.

Todos estos procedimientos nos van a tomar $O(n)$, ya que obtener las intersecciones entre el poligonos y las líneas horizontales las podemos obtener en $O(n)$ al recorrer todas las aristas, después podemos ver que en la pila que usemos solo vamos agregar a lo más una vez cada arista formada entre las intersecciones, las cuales van a ser $O(n)$ ya que como todas las intersecciones están sobre una línea horizontal tenemos que solo pueden hacer arista con la intersección a la izquierda o a la derecha, y de la pila a lo más vamos a eliminar cada arista que tenemos una vez, y además cómo este proceso se hace cuatro veces entonces tendremos que sigue respetando $O(n)$.

Notemos que este algoritmo no lo explicamos tanto y no fue tan detallado, esto debido a que en la exposición no se detallo tanto este algoritmo (fue algo rápido) y además creemos que el siguiente algoritmo es el importante ya que genera el polígono de visibilidad.

Si el polígono no tiene winding o ya se lo quitamos entonces vamos a realizar el segundo algoritmo de la presentación, que es el algoritmo para generar el polígono de visibilidad.

Usaremos una pila a la cual llamaremos S, en la pila vamos a ir guardando los vértices que forman parte del polígono de visibilidad, notemos que será el polígono de visibilidad de los vértices que hemos procesado.

Lo primero que hacemos es trazar una línea horizontal hacia la derecha del punto x, sea v_0 el punto más cercano a q donde se intersecta la línea horizontal y alguna arista del polígono, es decir trazamos la línea horizontal y vamos recorriendo todas las aristas del polígono para ver si se intersectan con la línea, si encontramos que se intersectan entonces vemos el punto de intersección y de todos esos puntos de intersección nos quedamos con el más cercano al punto x, entonces agregamos al punto v_0 al polígono.

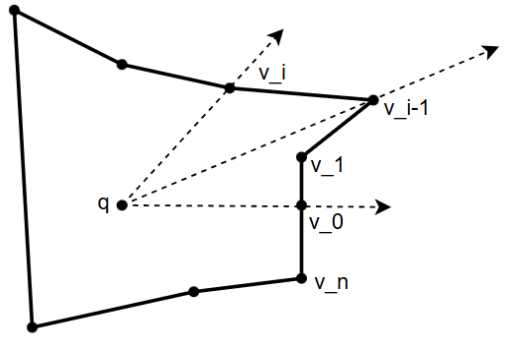
Ahora vamos a recorrer a los puntos del polígono en el sentido contrario a las manecillas del reloj, partiendo desde v_0 (tendremos que $i = 0$ al inicio), notemos que si estamos en el vértice v_i entonces el vértice siguiente (en el polígono) es v_{i+1} y el anterior es v_{i-1}.

Lo primero que hacemos es meter al vértice v_0 a la pila e inicializamos la variable i en 1 (esta variable nos sirve para saber en qué vértice vamos).

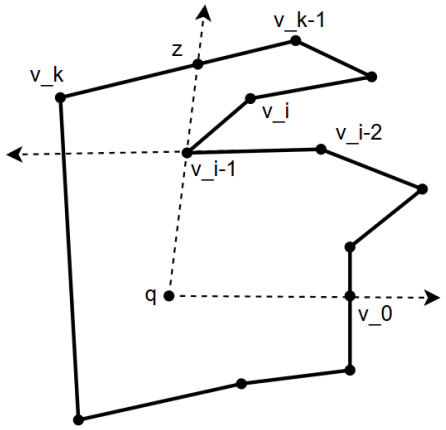
Para procesar los vértices que vamos recorriendo realizamos estos pasos, veremos cómo procesar al vértice v_i.

Paso 1, insertamos al vértice v_i a la pila, aumentamos i en uno y si se cumple que $i = n + 1$ entonces pasamos al Paso 10.

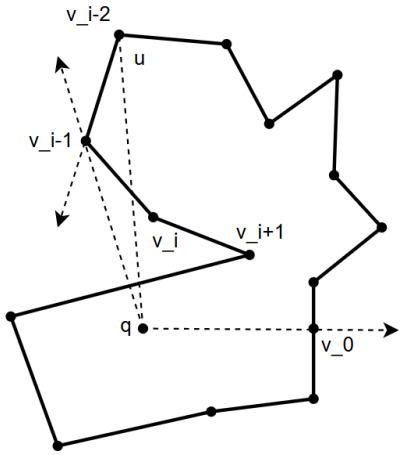
Paso 2, revisamos si v_i está a la izquierda de la línea que parte de q y pasa por v_{i-1}, si se cumple entonces pasamos al Paso 1, esto ya que los vértices en la pila están ordenados dependiendo de su ángulo con respecto al punto x (esto se debe a la forma en la que se insertan los vértices y que la pila representa nuestro polígono de visibilidad actual) y tenemos que el vértice v_i está a la izquierda por lo tanto sería el más grande de los vértices que tenemos en la pila y por lo tanto agregarlo respeta el orden.



Paso 3, revisamos si v_i está a la derecha de la línea que inicia en x y pasa por v_{i-1} y de la línea que inicia en v_{i-2} y pasar por v_{i-1}, si se cumple entonces recorremos los vértices en sentido contrarreloj empezando por v_{i+1} hasta encontrar un vértice v_k tal que cumple que el segmento de recta formado del vértice v_{k-1} al vértice v_k intersecta a la línea que inicia en q y pasa por v_{i-1}, al punto de intersección lo llamamos z, lo agregamos a la pila y actualizamos el valor de i para que ahora sea k ($i = k$), por último pasamos al Paso 1, esto ya que se cumple que v_i está a la derecha de la línea que inicia en x y pasa por v_{i-1} significa que el vértice v_i está a la derecha de lo que llevamos en la pila y por lo tanto si lo agregamos al tope de la pila ya no tendríamos el orden angular en la pila y como también se cumple que v_i está a la derecha de la línea que inicia en v_{i-2} y pasar por v_{i-1} entonces no podemos ver al punto v_i desde el punto q y por lo tanto se va agregar un vértice al que sí podamos ver, el cual es z ya que este vértice pasa por la línea que inicia en q y pasa por v_{i-1} así que el orden angular se va a respetar y será el siguiente punto que puede ver el punto q, ya que todos los vertices del poligono entre z y v_{i-1} no los podremos ver, ya que nuestro polígono ya no tiene windings.

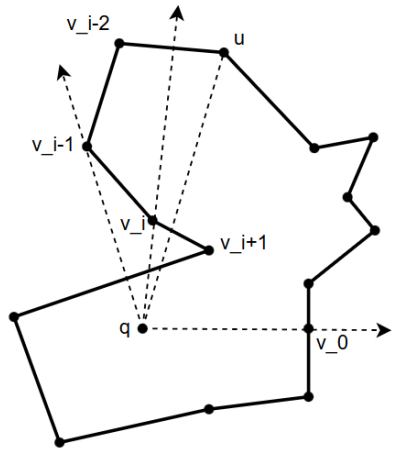


Paso 4, revisamos si v_i está a la derecha de la línea que inicia en x y pasa por v_{i-1} y a la izquierda de la línea que inicia en v_{i-2} y pasar por v_{i-1}, si se cumple entonces llamamos al vértice en el tope de la pila como u, luego sacamos al elemento en el tope de la pila (pop a la pila), ahora lo siguiente que vamos a hacer lo llamaremos Paso 4.1, mientras u sea un vértice del polígono P y el segmento de recta formado del vértice v_{i-1} al vértice v_i intersecte al segmento de recta formado del vértice q al vértice u vamos a sacar el tope de la pila, esto ya que se cumple que v_i está a la derecha de la línea que inicia en x y pasa por v_{i-1} significa que el vértice v_i está a la derecha de lo que llevamos en la pila y por lo tanto si lo agregamos al tope de la pila ya no tendríamos el orden angular en la pila y como también se cumple que v_i está a la izquierda de la línea que inicia en v_{i-2} y pasar por v_{i-1} entonces es posible que podamos ver al punto v_i desde el punto q (o al menos está más cercano de algunas partes del polígono de visibilidad que llevamos) y por lo tanto tenemos que ir sacando los vértices que no podemos ver (ya que el segmento de recta del vértice v_i al vértice v_{i-1} nos tapa la vista) de la pila.

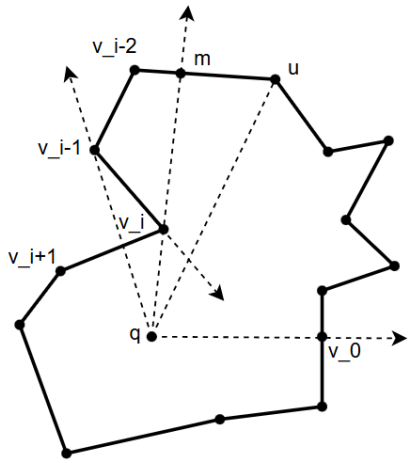


Paso 5, revisamos si el segmento de recta formado del vértice v_{i-1} al vértice v_i intersecta al segmento de recta formado del vértice q al vértice u, si esto se cumple

entonces ahora revisamos si se cumple que v_{i+1} este a la derecha de la línea que inicia en q y pasar por el vértice v_i, si se cumple actualizamos i aumentando uno y pasamos al Paso 4.1, esto ya que necesitamos seguir quitando vértices que no podemos ver de nuestra pila, pero ahora quitamos los que no podemos ver por el segmento de recta del vértice v_i al vértice v_{i+1}.

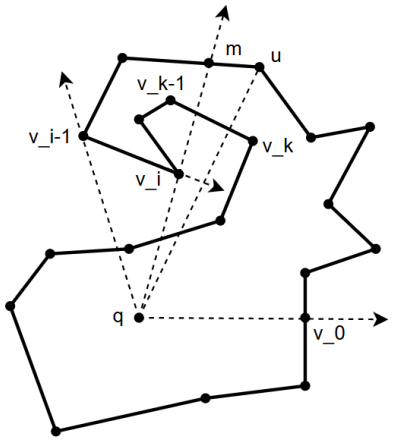


Paso 6, ahora calculamos al punto m que es el punto de intersección entre la línea que inicia en q y pasa por el vértice v_i y la arista que contiene al vértice u, cuando tengamos a m vamos a revisar si v_{i+1} está a la derecha de la línea que inicia en v_{i-1} y pasa por v_i, si esto se cumple entonces metemos a m a la pila y pasamos al Paso 1, esto ya que como se cumple que v_{i+1} este a la izquierda de la línea que inicia en q y pasar por el vértice v_i entonces tendremos que ya quitamos a todos los vértices que no podemos ver de la pila, pero ahora necesitamos agregar el último punto de la arista que contiene a u (que es m), ya que es el último vértice que podemos ver antes de que el vértice v_i nos tape los otros vértices que sacamos.

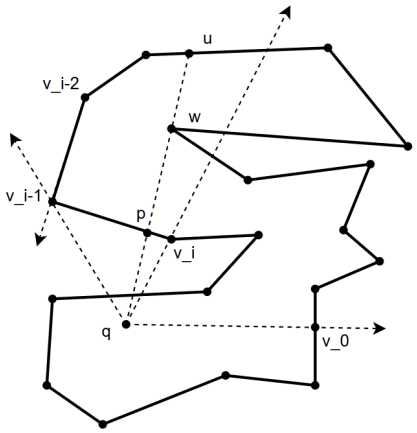


Paso 7, vamos a revisar a partir de v_{i+1} en sentido contrarreloj hasta encontrar un vértice v_k tal que cumpla con que el segmento de recta del vértice v_{k-1} al vértice v_k intersecta al segmento de recta del vértice m al vértice v_i, cuando tengamos al

vértice v_k entonces actualizamos el valor de i para que sea k ($i = k$) y pasamos al Paso 4.1, esto ya que como se cumple que v_{i+1} este a la izquierda de la línea que inicia en v_{i-1} y pasa por v_i entonces vamos a tener que el vértice m que encontramos no va a ser visible desde q, por lo tanto tenemos que buscar un nuevo vértice que su arista tapa la visión entre m y q, para volver a quitar los vértices de la pila los cuales son tapados por esa arista ($v_{k-1}v_k$).

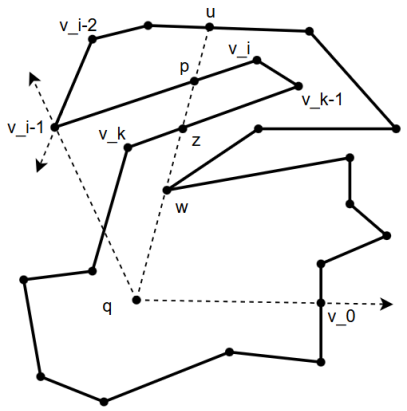


Paso 8, tendremos que el vértice w va a el vértice abajo de u (el segundo vértice al tope de la pila) y el vértice p será la intersección entre el segmento de recta del vértice v_{i-1} al vértice v_i y el segmento de recta del vértice u al vértice q, ahora revisamos si p está en el segmento de línea del vértice q al vértice w o que los vértices q, w y u no son colineales, si algo de esto se cumple entonces quitamos el tope de la pila y pasamos al Paso 4.1, es ya que vamos a tener que u no es un vértice del polígono P y fue agregado en algún paso anterior, entonces si tenemos que p está el segmento de línea del vértice q al vértice w vamos a tener que la visibilidad de los puntos w y u nos va ser bloqueada por la arista v_iv_{i-1}, por lo tanto tendremos que sacar los vértices que no podemos ver por esa arista.

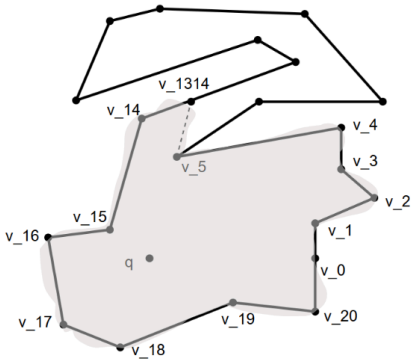


Paso 9, vamos a revisar a partir de v_{i+1} en sentido contrarreloj hasta encontrar un vértice v_k tal que se cumpla con que el segmento de recta del vértice v_{k-1} al vértice

v_k intersecta al segmento de recta del vértice w al vértice p, entonces llamaremos al punto de intersección de cómo z, insertamos a z en la pila, actualizamos el valor de i para que sea k ($i = k$) y pasamos al Paso 1, es ya que ahora p no está el segmento de línea del vértice q al vértice w por lo tanto vamos a tener que los vértices u y p no van a ser visibles por q, así que tenemos que buscar la arista que nos bloquea la vista para poder agregar la intersección entre esa arista y el segmento de línea de q a p, ya que esa intersección sería un punto el cual posiblemente podamos ver y nos bloquea la vista de p, y luego seguimos el proceso saltandonos los vértices entre v_i y v_{k-1} ya que no los vamos a poder observar desde q.



Paso 10, sacamos todos los vértices que tenemos en la pila y esos vértices forman el polígono de visibilidad.



Notemos que este algoritmo funciona ya que en la pila siempre tenemos que los vértices respetan el orden angular entre ellos de esta manera nuestro polígono de visibilidad se van formando bien y cuando llegamos a casos donde el orden se afectaría tenemos todos los casos para arreglar la pila (y por lo tanto el polígono) dependiendo de cuántos vértices de la pila no los podemos observar y cuantas vertices del poligono podemos saltar. Y de esta manera tendremos que la pila tendrá el polígono de visibilidad de los vértices que hemos procesado (suponiendo que ninguno de los vértices que nos faltan procesar nos tapan la visibilidad).
Ahora revisemos la complejidad de este algoritmo, al inicio agregamos un vértice v_0 el cual lo podemos encontrar en tiempo $O(n)$, ya que solo recorremos las aristas del polígono (que son $O(n)$ hasta encontrar la intersección, y luego lo agregamos que

también es $O(n)$ (insertar en una lista), luego se van a procesar los vertices del poligono, por lo tanto vamos a tener que cada vértice del polígono va entrar a la pila a lo más una vez y por lo tanto solo podrá salir de la pila una vez, además cuando revisamos los vértices a partir de uno para encontrar a un vértice que cumple algo, en algunos casos agregamos un vértice y vamos a lograr saltarnos los vértices entre el que estábamos procesando y el que encontramos, por lo tanto al final todos los vértices solo son procesados una vez, entonces como a lo más vamos a insertar $O(n)$ elementos a la pila y la inserción nos toma $O(1)$, además a lo más vamos a eliminar $O(n)$ elementos de la pila (la eliminación nos toma $O(1)$) y como todas las operaciones del proceso de los vértices como ver la dirección de un punto contra un segmento y ver si dos segmentos se intersectan nos toma $O(1)$, vamos a tener que todo este proceso nos va tomar $O(n)$, ya que cada vértice lo procesamos a lo más una vez y por lo tanto solo recorremos el límite del polígono una vez, y por último sacar los vértices que forman el polígono de visibilidad también nos toma $O(n)$ (tendrá a lo más $O(n)$ vértices), así que todo este algoritmo nos toma $O(n)$, todo gracias a que pasamos por cada vértice una vez, insertando cada vértice a lo más una vez y sacándolo a lo más una vez, además ya que los vértices creados (no eran del polígono originalmente) serán $O(n)$ ya que a lo más en el procesamiento de cada vértice agregamos uno y de esta manera todo nos queda en $O(n)$, tomando en cuenta que el polígono no tiene winding.

Y los dos algoritmos vimos que nos toman $O(n)$, por lo tanto obtener el polígono de visibilidad lo podemos hacer en $O(n)$.

Ejercicio 4

Árboles kd

Recordemos que los nodos internos del árbol representan los cortes que se realizan para ir partiendo el conjunto de puntos, notemos que dependiendo del valor de d entonces tendremos la cantidad de cortes (por ejemplo en $d = 2$ tendremos cortes horizontales y verticales), es decir, los nodos internos van a tener una línea que divide el subplano de una dimensión, por lo tanto supondremos que cada nodo interno sabe la dimensión la cual "divide", esto lo podemos ver como que si tenemos las dimensiones a_1, a_2, …, a_d entonces la raíz del árbol tendrá un corte o línea en la dimensión a_1, los hijos de la raíz tendrán cortes en la dimensión a_2 y así se sigue, por lo tanto podemos usar la profundidad del nodo interno para saber en qué dimensión hace el corte.

Inserción de un punto

Observamos que omitiremos el caso cuando el árbol esté vacío, ya que aquí solo creamos un nuevo nodo relacionado al punto y lo ponemos de raíz del árbol.

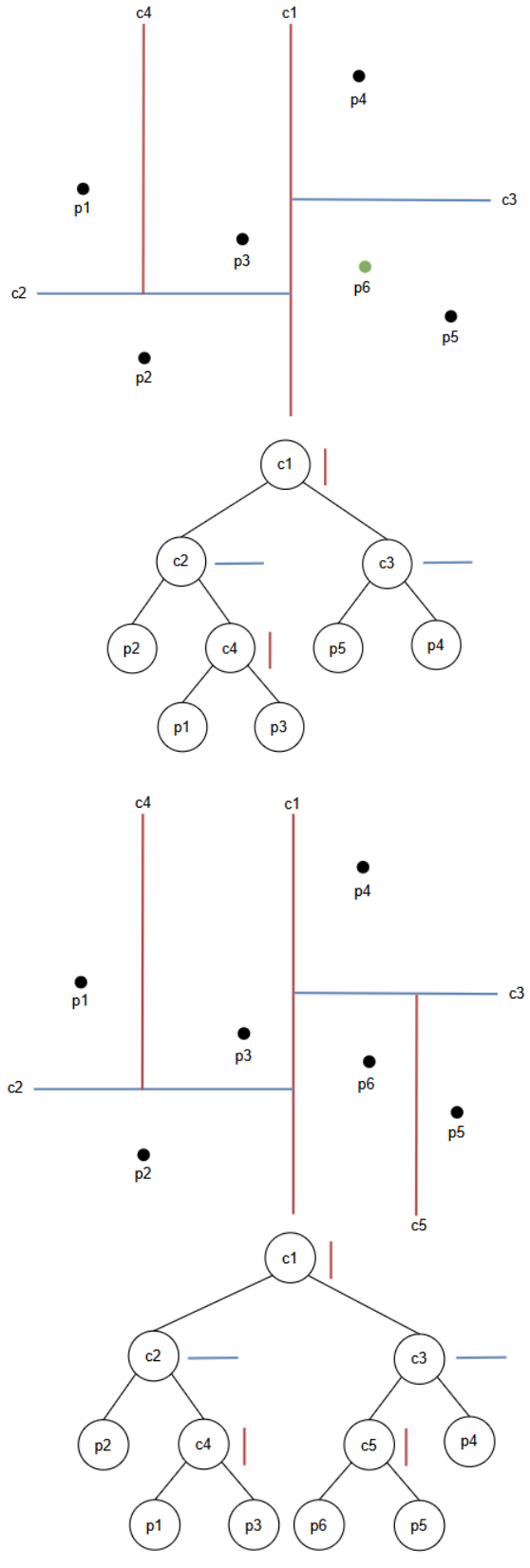
Lo primero que hacemos en este algoritmo es encontrar es encontrar la región que donde se va a encontrar el punto que queremos agregar, empezamos en la raíz del árbol y vamos a procesar los nodos internos hasta llegar a una raíz, para los nodos internos los procesamos de la siguiente manera, obtenemos la dimensión que divide el nodo, digamos que es la dimensión a_i, entonces vamos a comparar la coordenada en la dimensión a_i del corte o línea del nodo con la coordenada en la dimensión a_i del punto, por ejemplo en $d = 2$ tendremos que si el nodo hace un corte horizontal entonces vamos a comparar la coordenada y del corte y la coordenada y del punto, entonces tendremos dos casos, el primero es que la coordenada del punto es menor o igual a la coordenada del corte/línea en este caso vamos a descender por el hijo izquierdo (es decir va a ser el nuevo nodo que vamos a procesar), notemos que esto puedo cambiar (que bajemos por el hijo derecho en vez del izquierdo) dependiendo de la forma en la que se construyó el árbol, el otro caso es cuando la coordenada del punto es mayor a la coordenada del corte/línea entonces vamos a descender por el hijo derecho, y para procesar los nodos que son hojas vamos a solo regresar el nodo y el punto que guarda.

Cuando encontremos al nodo hoja llamaremos al punto que representa como c y para facilitar las cosas llamaremos al punto que vamos a insertar como g, ahora obtenemos al padre del nodo hoja, el cual es un nodo interno y por lo tanto obtenemos la dimensión la cual divide, digamos que la dimensión es a_m, ahora recordemos que un nodo interno debe tener dos hijos ya que cada nodo interno representa una división y recordemos que en clase vimos que cada región creada al final solo tendrá un punto (que son las hojas) por lo tanto no hay regiones vacías, entonces vamos a agregar un nuevo nodo interno el cual tomará el lugar del nodo hoja que encontramos, pero ahora veamos cual será el corte y dimensión del nuevo

nodo interno, al cual llamaremos f, sabemos que la dimensión del nodo padre es a_m entonces la dimensión de f sera a_{m+1} (notemos que si $m + 1 > d$, entonces será a_1) esto ya que vamos a hacer un corte en la siguiente dimensión para que se cumple que con la profundidad del nodo podemos saber en qué dimensión es el corte, y para el corte vamos obtener las coordenadas de c y g en la dimensión a_{m+1}, luego sacamos el valor s el cual está en el valor de las coordenadas de c y g en la dimensión a_{m+1} (es decir, el que está entre esos dos valores), por lo tanto el corte/línea del nuevo nodo f va ser una línea en la dimensión a_{m+1} sobre el valor s en esa coordenada (es decir es una línea que tiene la coordenada a_{m+1} con el valor fijo s), entonces agregamos al nodo f en la posición donde estaba el nodo hijo que encontramos (es decir si el nodo hijo era un hijo izquierdo del nodo padre entonces ahora f será el hijo izquierdo del nodo padre, análogamente para si es hijo derecho), ahora vamos a revisar la coordenada de c en la dimensión a_{m+1} y la comparamos con s, si la coordenada de c es menor o igual entonces pondremos al nodo que representa a c como hijo izquierdo de f y creamos un nodo para que represente a g (el punto a agregar) y lo ponemos como hijo derecho de f, en caso de que a coordenada de c es mayor entonces pondremos al nodo que representa a c como hijo derecho de f y creamos un nodo para que represente a g (el punto a agregar) y lo ponemos como hijo izquierdo de f.

De esta manera insertamos al nuevo punto, pero notemos que el balance del árbol puede ser afectado, por lo tanto las consultas o encontrar puntos en el árbol pueden tener mayor complejidad.

En el siguiente ejemplo se agrega el punto p6 y el nodo hoja encontrado es el nodo del punto p5.



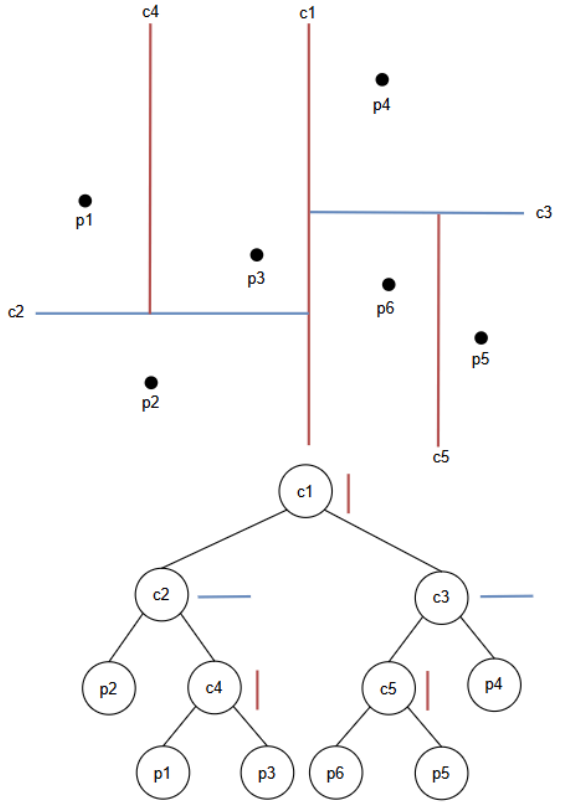
Notemos que esto funciona ya que recorremos nuestro árbol para encontrar la región que debe contener al nuevo punto (es la región donde se encuentra el nodo hoja que encontramos) y luego esa región la dividimos en dos mediante un corte

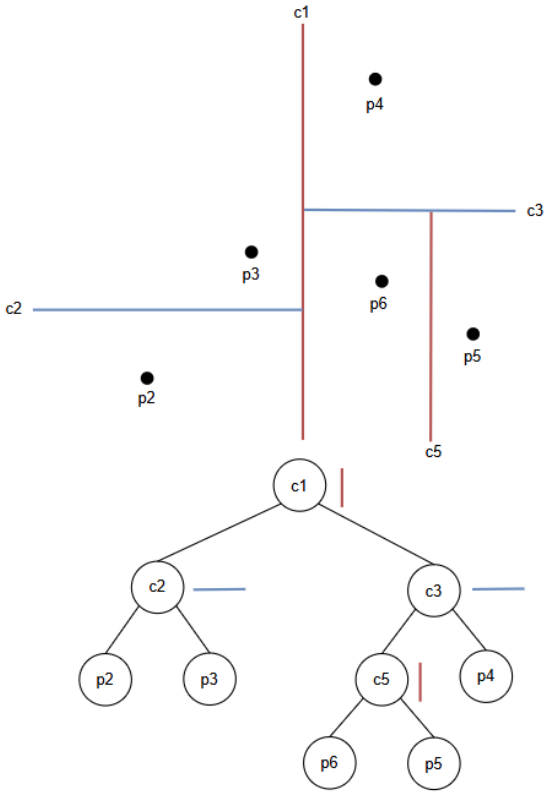
(que es el nuevo nodo interno f que agregamos) para que de esta forma cada región solo contenga un punto.
Por último veamos la complejidad de esto, recordemos que en la clase del 07/03/25 vimos que la complejidad de consulta es $O(n^{\frac{1}{d}} + k)$, donde k es la cantidad de puntos en el rango, ahora notemos que la forma en la que encontramos el nodo hoja es la misma a la vista en clase solo que ahora nuestro rango de búsqueda sólo es un punto por lo tanto $k = 1$ y la complejidad de ir bajando por el árbol debe ser la misma, entonces tendremos que encontrar al nodo hoja nos toma $O(n^{\frac{1}{d}})$, luego obtener a su padre lo podemos hacer en $O(1)$ (si es que el árbol tiene que cada nodo tiene referencia a su padre), y por último obtener la nueva dimensión del nuevo nodo f lo podemos hacer en $O(1)$ ya que usamos la dimensión del nodo padre para calcularla (y solo son sumas), luego notemos que reemplazamos el nodo hoja por el nuevo nodo f, lo cual podemos hacerlo en $O(1)$ ya que solo es cambio de referencias, luego hacemos la línea/corte del nodo f lo cual también es $O(1)$ (solo es hacer una línea y obtener un valor entre dos valores) y por último comparamos dos coordenadas y agregamos dos hijos al nodo f, lo cual lo podemos hacer en $O(1)$ ya que sólo apuntamos referencias a los hijos, por lo tanto podemos ver que todo el proceso nos va a tomar $O(n^{\frac{1}{d}})$.

Eliminación de un punto
Para este proceso vamos a realizar algo muy similar a la forma en la que agregamos un punto.
Lo primero que hacemos es buscar si el punto a eliminar, digamos que es j, está en el árbol, para esto vamos a recorrer y buscar sobre el árbol de igual manera en la que encontramos al nodo hoja en la inserción de un punto solo que ahora cambia la forma de procesar a un nodo hoja, lo que cambia es que en vez de regresar primero comparamos si las coordenadas del punto relacionado al nodo hoja son las mismas que del punto j, si no son iguales entonces terminamos la eliminación (ya que el punto no está en el árbol) pero si son iguales entonces regresamos al nodo hoja, al cual llamaremos r.
Observemos que si r es la raíz del árbol entonces solo devolvemos al árbol vacío y terminamos.
Cuando tengamos al nodo hoja r, vamos a obtener a su padre al cual llamamos t, después vamos a obtener al otro hijo de t y lo llamamos w, después vamos a obtener al nodo padre de t y lo llamamos q, ahora tendremos dos casos uno donde w (el hermano) es una nodo hoja y otro cuando es un nodo interno, veamos que vamos hacer cuando es un nodo hoja, agregamos (o ponemos) a w en la posición donde estaba t (es decir si t era un hijo izquierdo de q entonces ahora w será el hijo izquierdo de q, análogamente para si es hijo derecho), posteriormente vamos a eliminar a los nodos t y r, y con eso terminamos la eliminación, ahora veamos qué hacer cuando w (el hermano) es un nodo interno, como no podemos ponerlo en el

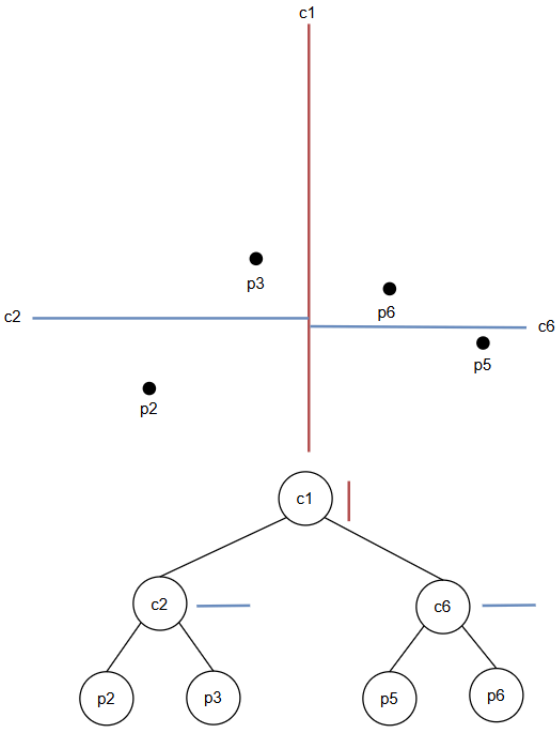
lugar de t (el padre) ya que esto nos causaría tener dos cortes en la misma dimensión uno al lado del otro en el árbol, por lo cual obtenemos a todos los puntos en el subárbol de w y los ordenamos, luego realizamos un corte en la dimensión de t tal que parte a la mitad a los puntos que ordenamos y agregamos un nodo interno con ese nuevo corte en el lugar de t, luego las mitades generadas por ese corte las volvemos a cortar con otros cortes pero ahora en la siguiente dimensión y agregamos los nodos de esos cortes, esto lo repetimos hasta llegar a que cada región solo tenga un vértice, es decir estamos reconstruyendo el subárbol del nodo hermano y poniéndolo en el lugar del nodo padre, esta manera nuestro árbol estará bien.

En el siguiente ejemplo (continuación del ejemplo anterior) se elimina el punto p1.





Y si luego eliminamos al punto p4 nos queda así.



Notemos que esto funciona ya que recorremos nuestro árbol para encontrar al punto a eliminar, luego obtenemos a su padre, el cual representa un corte, y a su hermano, el cual cuando representa otro punto, al hacer que el hermano tome el lugar del padre lo que estamos haciendo es unir las regiones del punto a eliminar y la del hermano, ya que al eliminar al padre se elimina el corte que las dividía y por lo tanto las regiones se unen, luego al quitar al nodo del punto a eliminar solo vamos a

dejar al hermano en la nueva region y por lo tanto se cumplira que cada region tiene un punto, pero cuando el hermano representa un corte entonces tenemos que volver a reconstruir el subárbol del hermano, ya que si no lo hacemos vamos a tener dos cortes en la misma dimensión seguidos.
Por último veamos la complejidad de esto, como vimos en el caso de la inserción tendremos que encontrar al nodo hoja nos toma $O(n^{\frac{1}{d}})$, ya que hacemos el mismo proceso para bajar por el árbol solo que ahora hacemos una última comparación con el punto del nodo hoja para ver si es el que tenemos que eliminar, luego obtener a su padre lo podemos hacer en $O(1)$ (como vimos antes) de igual manera podemos obtener al hermano y abuelo en $O(1)$, cuando el hermano es una hoja entonces reemplazamos el nodo hermano por el nodo padre lo podemos hacer en $O(1)$ ya que solo es cambio de referencias (ya que el hermano toma el lugar del padre), luego solo eliminamos al nodo hijo que encontramos y al nodo padre lo cual podemos hacer en $O(1)$ (solo es quitar las referencias que apuntan hacia el nodo padre del nodo hermano y del nodo abuelo, pero esto ya se debería haber hecho cuando el hermano toma el lugar del padre), pero cuando el hermano es un nodo interno vamos a tener que reconstruir su subárbol, por lo cual tenemos que ordenar los puntos de este subárbol y luego ir haciendo cortes hasta dejar regiones con un solo punto, recordemos que construir un árbol kd nos toma $O(d\,n\,log\,n)$ (lo vimos en la clase del 07/03/25) por lo tanto podemos reconstruir el árbol del nodo hermano en $O(d\,n\,log\,n)$, por lo tanto podemos ver que todo el proceso nos va a tomar $O(n^{\frac{1}{d}} + d\,n\,log\,n)$.

Árboles de rangos
Inserción de un punto
Observamos que omitiremos el caso cuando el árbol esté vacío, ya que aquí solo creamos un nuevo nodo relacionado al punto y lo ponemos de raíz del árbol.
Antes de ver todo vamos a recordar como hacer una inserción en un Binary Search Tree donde las hojas serán los nodos que tendrán los puntos, esto debido a que como los árboles usados en el árbol de rangos son BST, entonces primero es ir bajando por el árbol usando la coordenada en la dimensión del árbol para hacer las comparaciones y saber por donde bajar hasta llegar a un nodo hoja, luego vemos si el nodo hoja que encontramos tiene hermano o no, si tiene hermano entonces creamos un nuevo nodo interno y lo ponemos en el lugar del nodo hoja que encontramos y ponemos como hijos al nuevo elemento a insertar y el nodo hoja (solo procuramos que el menor sea el hijo izquierdo), pero cuando el nodo hoja no tiene hermano entonces revisamos si el nodo hoja es el hijo izquierdo o derecho, si es el hijo izquierdo entonces revisamos si el nuevo elemento es menor o igual al punto del nodo hoja (esto comparando la coordenada de la dimensión del árbol), si es mayor entonces ponemos al nuevo elemento como hermano del nodo hoja (es decir como hijo derecho), pero si es menor entonces creamos un nuevo nodo

interno el cual ponemos en el lugar del nodo hoja y luego ponemos al nuevo elemento como hijo izquierdo y al nodo hoja como hijo derecho, luego de agregar al nuevo elemento vamos a tener que ir subiendo por los nodos padres para poder ir actualizando que los nodos internos del árbol tengan al más grande del subarbol izquierdo, esto lo hacemos si subimos por un hijo izquierdo entonces revisamos el valor actual del padre y si el valor del nuevo elemento agregado es mayor entonces actualizamos el valor pero si no entonces seguimos subiendo hasta llegar a la raiz del arbol, de esta manera insertamos en el árbol BST.

Ahora veamos cómo agregar un nuevo punto a nuestro árbol de rangos, lo primero que vamos a hacer es en el árbol de la primera dimensión vamos a insertar al nuevo elemento de la forma que explicamos como insertar en un árbol BST, pero antes de ir subiendo vamos a notar unas cosas, si durante la insercion agregamos un nuevo nodo interno entonces cuando lleguemos a este nuevo mientras subimos vamos a tener que crear un arbol nuevo en la siguente dimension con los elementos que tiene como hijos (es decir creamos un nuevo arbol con la base canonica de ese nodo interno pero ahora usando la coordenada de la siguente dimension para hacer las coomparaciones), notemos que como sera crear un arbol de dos elementos (el nodo hoja y el nuevo que agregamos) esto se puede hacer facilmente, y luego de crear este nuevo arbol si hay tenemos más dimensiones entonces creamos otro arbol, entonces creamos suficientes arboles como dimensiones nos queden, luego mientras vamos subiendo por los nodos internos vamos a tener que cada nodo interno tiene un arbol en la siguente dimension por lo tanto entramos a ese arbol e insertamos al nuevo elemento y repetimos todo este proceso otra vez (el de agregar el nuevo elemento, crear nuevos arboles si agregamos un nodo interno e ir subiendo por el arbol entrenando a los arboles de otras dimensiones de los nodos internos para agregar el nuevo elemento), asi vamos subiendo por los nodos internos hasta llegar a la raiz del arbol de la primera dimension y terminar, en resumen lo que hacemos es insertamos normalmente en el árbol de la primera dimension, y si durante la insercion agregamos un nodo interno entonces creamos sus arboles de las siguientes dimensiones, luego vamos subiendo por los nodos internos padres hasta llegar a la raiz, en cada nodo interno visitado actualizamos para tener al más grande del subarbol izquierdo y entramos al arbol de la siguiuente dimension relacionado al nodo interno y repetimos todo este proceso de insertar el nuevo elemento y subir hasta la raíz.

Notemos que esto funciona ya que primero insertamos en el árbol de la primera dimensión y luego vamos subiendo hasta la raíz pasando por todos los árboles de las demás dimensiones relacionados a los nodos internos que pasemos para insertar el nuevo elemento, esto es bueno ya que pasamos por todos los árboles que van a tener al nuevo elemento y por lo tanto tenemos que insertarlo en cada uno de estos árboles para que el nuevo elemento este en todos los árboles en los que tiene que estar.

Ahora veamos la complejidad de esto, lo primero que tenemos que notar es que para encontrar el lugar donde vamos a insertar al nuevo elemento tenemos que bajar por el árbol lo cual nos toma su altura así que nos tomara $O(log\ n)$ bajar por

cada árbol, luego insertar al nuevo elemento lo podemos hacer en tiempo $O(1)$, después tenemos que ir subiendo por el árbol lo cual también nos toma $O(log\, n)$ (la altura del árbol) pero aquí es donde entramos a otros árboles en otras dimensiones donde entramos a uno por cada nodo interno en nuestro camino los cuales son $O(log\, n)$, ahora notemos que si estamos subiendo por un árbol de la primera dimensión entonces entramos a árboles de la segunda dimensión donde vamos a tener que volver a insertar, y si estamos en un árbol de la i dimensión entramos a árboles de la dimensión $i + 1$, esto lo vamos seguir haciendo hasta que en los árboles de la dimensión d no entremos a más árboles, por lo tanto todo esto en conjunto podemos ver que vamos a entrar a muchos árboles pero en total todos los árboles que visitamos van a ser algo igual a la cantidad que visitamos en una consulta, por lo tanto nos va a tomar $O(log^d n)$, ya que de los $O(log\, n)$ nodos internos de la primera dimensión entramos a un árbol de la segunda dimensión en cada uno, luego en cada nodo interno de los árboles de la segundo entramos a un árbol de la tercera dimensión, así hasta llegar a árboles de la tercera dimensión d, y como todos los árboles tendrán altura a lo más $O(log\, n)$ podemos ver que todo nos tomara $O(log^d n)$, pero recordemos que cuando en la inserción agregamos un nuevo nodo interno tenemos que crear sus árboles en las demás dimensiones usando su base canónica por lo tanto crear cada árbol nos toma $O(1)$ ya que solo tendrán dos elementos y cómo creamos a lo más d nuevos árboles entonces en total toda la inserción nos tomara $O(log^d n + d)$.

Eliminación de un punto

Observamos que omitiremos el caso cuando el árbol no contiene al punto por eliminar, ya que aquí solo tenemos que buscar si el punto está en el árbol de la primera dimensión y si no lo encontramos entonces terminamos.

Antes de ver todo vamos a recordar como hacer una eliminación en un Binary Search Tree donde las hojas serán los nodos que tendrán los puntos, entonces primero vamos bajando por el árbol usando la coordenada en la dimensión del árbol para hacer las comparaciones y saber por donde bajar hasta llegar a un nodo hoja, cuando lleguemos al nodo hoja vamos a revisar si el punto relacionado al nodo hoja es el mismo, si no lo son entonces terminamos pero si son el mismo entonces vamos a revisar si el nodo hoja tiene un hermano, si tiene un hermano entonces solo borramos el nodo hoja (que está relacionado al punto por eliminar) pero si no tiene un hermano entonces borramos el nodo hoja y al padre del nodo hoja (ya que si no hacemos esto el padre se volviera un nodo hoja pero no esta relacionado a ningun punto), notemos que esta eliminacion es suponiendo que el arbol al inicio esta balanaceado ya que en otro caso se dejarian nuevos nodos hojas (que antes eran nodos internos) los cuales son estan relacionados a ningun punto, para resolver esto podemos ir subiendo por lo nodos padres hasta encontrar uno con dos hijos y borrar todos los demas, luego de eliminar al elemento vamos a tener que ir subiendo por los nodos padres para poder ir actualizando que los nodos internos del

árbol tengan al más grande del subarbol izquierdo, esto lo hacemos comparando el elemento que eliminamos y los valores que tendran los hijos del nodo que vamos a actualizar o revisar, de esta manera vamos subiendo hasta llegar a la raiz del arbol, de esta manera eliminamos en el árbol BST.
Ahora veamos cómo agregar un nuevo punto a nuestro árbol de rangos, lo primero que vamos a hacer es en el árbol de la primera dimensión vamos a eliminar al punto de la forma que explicamos cómo eliminar en un árbol BST, pero cuando estemos subiendo por los nodos padres para llegar a la raíz vamos a hacer algo similar en la inserción, lo que vamos a hacer es que si estamos en un nodo interno entonces entramos al árbol de la siguiente dimensión (si es que existe) relacionado a ese nodo interno (es decir que se creó con su base canónica), luego de entrar al árbol de la siguiente dimensiona vamos a eliminar al punto de ese árbol y repetir todo el proceso de subir hasta la raíz arreglando los valores de los nodos internos y entrando a los árboles de las siguiente dimensiones para eliminar el punto y repetir todo esto, entonces lo que hacemos es entrando a todos los árboles (de todas las dimensiones) de todos los nodos internos por los cuales pasamos mientras intentamos llegar a la raíz luego de la eliminación en un árbol.
Notemos que esto funciona debido a que al inicio eliminamos el punto del árbol de la primera dimensión y después vamos subiendo hasta la raíz pasando por nodos internos en los cuales entramos a los árboles de las demás dimensiones para eliminar al punto en esos árboles y entrar a los demás árboles relacionados a los nodos internos que encontramos al subir a la raíz, y todo esto nos sirve ya que pasamos o visitamos todos los árboles que tienen al elemento que vamos a eliminar y por lo tanto eliminamos al punto de todos los árboles (de todas la dimensiones) que lo tienen, logrando eliminar el punto del árbol de rangos.
Ahora veamos la complejidad, recordemos que encontrar el elemento que vamos a eliminar lo hacemos bajando por el árbol lo cual nos tomará su altura la cual es $O(log\, n)$, después para eliminar al ndoo relacionado al punto a eliminar lo podemos hacer en tiempo $O(1)$ ya que solo es borrar nodos y actualizar referencias, después tenemos que ir subiendo por los nodos internos hasta llegar a la raíz del árbol lo cual nos tomará $O(log\, n)$ (ya que es similar a bajar por el árbol) pero mientras vamos subiendo por el árbol tenemos que entrar a los árboles de las otras dimensiones y como entramos a uno por cada nodo interno que usamos para subir vamos a tener que en un árbol vamos a tener $O(log\, n)$ nodos internos, entonces al subir por el árbol de la primera dimensión vamos a entrar a $O(log\, n)$ árboles de la segunda dimensión, luego en cada árbol de la segunda dimensión vamos a entrar a $O(log\, n)$ árboles de la tercera dimensión, y de esta manera entramos a los árboles de la siguiente dimensión hasta entrar a árboles de la dimensión d y cuando subamos estos árboles ya no vamos a entrar a más árboles, entonces podemos notar que al final vamos a entrar a varios árboles a los cuales vamos a tener que eliminar un elemento de cada uno y como entramos a los árboles de las dimensiones 1 a d vamos tener que todo esto nos va a tomar $O(log^{d}\, n)$ esto debido a que los árboles tienen $O(log\, n)$ nodos internos los cuales vamos a tener que

recorrer bajando y subiendo, y cada vez que entremos a árboles de otra dimensión los árboles que llevamos se multiplican por $O(log\ n)$, entonces al final todo esto nos toma $O(log^d\ n)$ (es similar a la complejidad de la inserción, ya que tenemos que bajar y subir por los árboles para agregar o eliminar un punto y luego entrar a los árboles de los nodos internos por los que pasemos).

Observemos que la complejidad de estos algoritmos es suponiendo que antes de realizar los algoritmos los árboles están balanceados, ya que si aplicamos varias veces estos algoritmos los árboles se van a desbalancear lo cual cambiará la complejidad de bajar por lo árboles y la altura de estos.
Notemos que otra alternativa a realizar estos algoritmos es volver a construir los árboles desde cero, pero con la eliminación o inserción en el conjunto de puntos.

Ejercicio 5

Usaremos contradicción para la demostración.
Pero antes de la demostración recordemos una propiedad de la gráfica/triangulación de Delaunay, la cual vimos en la clase del 31/03/25 y en la ayudantía del 20/05/25, lo primero es que la gráfica de Delaunay es una triangulación excepto cuando los puntos están alineados o cuando 4 o más círculos son cocirculares, ahora la otra propiedad es que la circunferencia circunscrita en cualquiera de los triángulos de la triangulación de Delaunay está vacía, es decir no hay ningún punto del conjunto del cual salió la triangulación de Delaunay.
Sea P un conjunto de n puntos en el plano, sea $D(P)$ la gráfica de Delaunay de P y sea $T(P)$ el árbol generador de peso mínimo euclidiano de P.
Ahora notemos una característica de la triangulación de Delaunay, veamos que dos puntos de P tienen una arista que los une en $T(P)$ si y sólo si existe un círculo vacío que pase por esos dos puntos.
Para esto supongamos que x y y serán los puntos de P, ahora veamos la iba, supongamos que la arista formada por x y y, la cual llamaremos z forma parte de $T(P)$, ahora como estos dos puntos tienen una arista (z) en $T(P)$ significa que las regiones de los puntos x y y en el Voronoi de P son vecinas y comparten una arista de Voronoi, la cual llamaremos w, ahora notemos que cualquier punto en w está a la misma distancia de x y y (esto ya que las regiones en Voronoi tienen los puntos más cercanos) y tambien cualquier punto en w tendrá como puntos más cercanos de P a x y y (igual por la manera en que se construye Voronoi, notemos que aquí puede existir un tercer punto más cercano por lo de la propiedad del círculo vacío pero esto no nos afecta, ya que el otro punto no se encontraría dentro del círculo), ahora notemos que si tomamos un círculo con centro en algún punto de w tal que ese circule pase por x y y tendremos que ese círculo es vacío, es decir ningún otro punto de P pasa por ahí, esto por las dos observaciones que hicimos antes (principalmente que x y y son los puntos de P más cercanos), por lo tanto existe un círculo vacío que pasa por x y y.
Veamos el regreso, ahora usaremos a los puntos u y v de P, supongamos que existe un círculo vacío, al cual llamaremos h, que pasa por u y v, ahora notemos que el centro de h, al cual llamaremos q, debe formar parte de la arista de Voronoi entre las regiones de u y v esto debido a que q está a la misma distancia de u y v (ya que es el centro del círculo) y los puntos más cercanos de P a q son u y v (ya que el círculo es vacío, notemos que puede que q tenga otro punto de P igual de cercano que u y v, por la propiedad del círculo vacío, pero no nos afecta eso), por lo cual tenemos que las regiones de Voronoi de u y v comparten una arista, por lo tanto estas regiones son vecinas y por la definición de $T(P)$ tendremos existe una arista entre u y v que está en $T(P)$, por lo tanto los dos puntos tienen una arista que los une en $T(P)$.
Ahora supongamos, para generar una contradicción, que existen dos puntos a y b en P tal es que la arista formado por a y b, la llamaremos c, forma parte de $T(P)$

pero no de $D(P)$, notemos que como c no forma parte de $D(P)$ tendremos que no cumple la característica que revisamos arriba, eso quiere decir que ningún círculo que pase por los puntos a y b estará vacío y por lo tanto tendrá al menos un punto de P, así que notemos que el círculo que tiene de diámetro a la arista c tendrá un punto de P dentro, a este punto lo llamaremos como d.
Ahora notemos que si quitamos la arista c de $T(P)$ entonces vamos a partir a $T(P)$ en dos subárboles, esto debido a que como el árbol generador de peso mínimo euclidiano es un árbol entonces no tiene ciclo.
Tendremos dos casos, cuando el punto d está en el subárbol que tiene al punto a y cuando d está en el subárbol que tiene al punto b.
Veremos el primer caso, cuando d está en el subárbol que tiene al punto a, notemos que si le quitamos la arista c a $T(P)$ y le agregamos la arista formada por los puntos b y d (notemos que no podemos agregar la arista formada por a y d, ya que generaría un ciclo en un subárbol generado al quitar la arista c) tendremos que se forma otro árbol generador, ya que los dos subárboles generados de quitar la arista c a $T(P)$ son unidos mediante la arista que va de b a d, llamaremos a esta arista como e para facilitar las cosas, y por lo tanto es un árbol generador, a este nuevo árbol generador lo llamaremos $S(P)$, ahora revisemos cuál será el peso de $S(P)$, para esto diremos que $W_{T(P)}$ es el peso de $T(P)$, W_c será el peso de la arista c y W_e será el peso de la arista e, entonces el peso de $S(P)$ ($W_{S(P)}$) va a ser $W_{S(P)} = W_{T(P)} - W_c + W_e$, esto debido a que quitamos la arista c y agregamos la arista e, ahora notemos que como la arista c es el diametro del circulo (el del inicio que no cumplia la propiedad del círculo vacío) entonces tendremos que el peso (euclidiano) de la arista c debe ser mayor que el peso de la arista e, debido a que el diámetro de un círculo (la arista c) es más grande que cualquier otro segmento dentro del círculo (la arista e), por lo tanto tendremos que $W_{T(P)} > W_{S(P)}$, notemos que esto significa que $W_{S(P)}$ sería el árbol generador de peso mínimo euclidiano de P y por lo tanto $W_{T(P)}$ no puede ser árbol generador de peso mínimo euclidiano de P, lo cual es una contradicción de que $T(P)$ el árbol generador de peso mínimo euclidiano de P, por lo tanto en este caso tendremos que c debe formar parte de $D(P)$.
Ahora veamos el caso cuando d está en el subárbol que tiene al punto b, este caso va a ser análogo al caso anterior solo que vamos a agregar la arista arista formada por los puntos a y d, y se llegará a la misma contradicción de que el árbol generador formado por quitar la arista c y agregar la arista del vértice a y d va a ser el árbol generador de peso mínimo euclidiano de P (ya que tendremos el mismo argumento de que la arista c al ser el diámetro del círculo entonces tendrá peso mayor a que la otra arista), y así generar una contradicción de que $T(P)$ el árbol generador de peso mínimo euclidiano de P, por lo tanto en este caso también tendremos que c debe formar parte de $D(P)$.

Por lo tanto vemos que en ambos casos llegamos a una contradicción por suponer que la arista c (que está en $T(P)$) no está en $D(P)$, por lo tanto podemos concluir que c debe estar en $D(P)$ y con esto podemos concluir que árbol generador de peso mínimo euclidiano de P en el plano es una subgráfica de Delaunay, esto debido a que ambos tienen todos los puntos de P y toda arista en el árbol generador de peso mínimo euclidiano de P debe estar en la gráfica/triangulación de Delaunay.

Ejercicio 6

El algoritmo de Kirkpatrick nos sirve para encontrar la cara que contiene a un punto dada una subdivisión plana de n puntos y un punto de consulta.
Veamos ahora los pasos del algoritmo:
Paso 1, se calcula el cierre convexo de los puntos de la subdivisión, sabemos que este paso lo podemos hacer en tiempo $O(n\ log\ n)$ por algoritmos vistos en clase.
Paso 2, encerramos todo el cierre convexo en un enorme triángulo, esto lo podemos hacer en tiempo $O(n)$, ya que tenemos que conocer los puntos a los extremos del cierre convexo.
Paso 3, se lanzan rayos de los vértices del triángulo grande a los vértices del cierre convexo, lo podemos hacer en tiempo $O(n)$, ya que a lo más el cierre convexo estará formado por n puntos.
Paso 4, se triangula la subdivisión con el cierre convexo, esto lo podemos hacer en $O(n\ log\ n)$, ya que a la subdivisión solo le agregamos tres puntos (los del triángulo grande) entonces tendríamos una subdivisión plana de $n + 3$ puntos y sabemos la complejidad para triangular eso.
Paso 5, se toman vértices de la subdivisión original, notemos que estos vértices tomados deben ser independientes entre ellos (no compartir aristas), y los quitamos (y por lo tanto también a las aristas que salen de ellos), de esta manera generando nuevas caras y esas caras nuevas las triangulamos, notemos que estas nuevas triangulaciones nos van a tomar $O(1)$ cada una, ya que triangulamos las nuevas caras formadas de las cuales sabemos qué vértices están conformadas y más adelante veremos que estas caras son formadas por a lo más 8 vértices debido a que los puntos seleccionados a quitar tienen a lo más grado 8, por lo tanto como quitamos a lo más $O(n)$ vértices y generamos a lo más $O(n)$ caras (una por vértice quitado), nos va a tomar en total $O(n)$ triangular a todas las caras nuevas.
Paso 6, repetimos el paso 5 hasta solo dejar el triángulo grande con nada adentro, notemos que tenemos que ir guardando las triangulaciones generadas luego del paso 4 y luego de cada iteración del paso 5.
Paso 7, usaremos una estructura de árbol para guardar las triangulaciones generadas (como tal se van a guardar caras o triángulos que genera la triangulación), tendremos que la raíz será el triángulo vacío y tendrá como hijos los tres triángulos (caras) que estaban en la penúltima iteración del paso 5, es decir las caras que genera la triangulación del triángulo grande y del último vértice quitado, vamos construyendo el árbol de manera que cada nodo (o triángulo/cara) tendrá como hijos a los triángulos o caras que lo componen en las demás triangulaciones y así tendremos que las hojas del árbol serán los triángulos o caras de la triangulación del paso 4.
Una cosa importante a notar de este árbol es que un nodo puede tener más de un papá, pero cada nodo del árbol tiene un número $O(1)$ de hijos, ya que el número de regiones generadas al quitar un vértice es a lo más su grado y por el lema visto en la clase del 28/03/25 que dice “Dada una subdivisión plana y conexa de n vértices

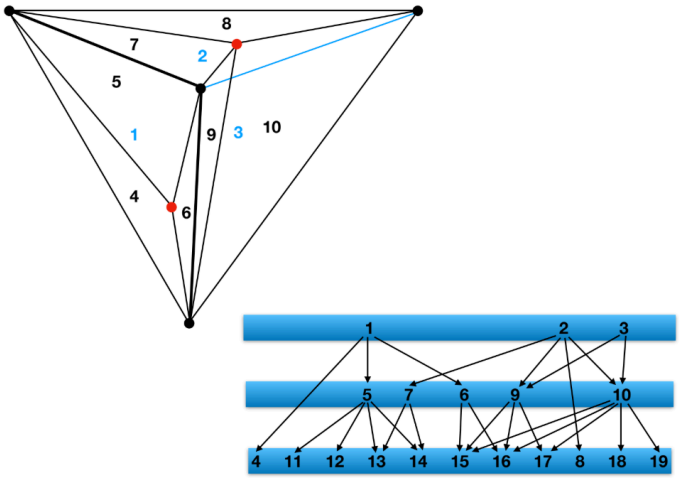
existe un conjunto independiente que contiene vértices de grado a lo más 8, con al menos $\frac{n}{18}$, y al elegir que vertices quitar se tiene que tener cuidado para elegir vértices no adyacentes y con grado no tan grande.

Por lo tanto vamos a tener que en la primera iteración del paso 5 vamos a quitar $\frac{n}{18}$ vértices, y para las demás iteraciones podemos verlo así:

Luego de la iteración 1 nos quedarán n_1 vértices de la subdivisión original, con $n_1 = n - \frac{n}{18}$, después de la iteración 2 nos quedarán n_2 vértices de la subdivisión original, con $n_2 = n_1 - \frac{n}{18} = (n - \frac{n}{18}) - \frac{n}{18}$, luego de la iteración 3 nos quedarán n_3 vértices de la subdivisión original, con $n_3 = n - \frac{n}{18}$, entonces de esta manera podemos ver que en cada etapa quitamos una fracción lineal de los vértices de la subdivisión, por lo tanto para quitarlos todos (las veces que repetiremos el paso 5) será $log\, n$ ($log_{18} n$), y con esto podemos saber también la altura que tendrá nuestro árbol, la cual es de $log\, n$, ya que cada vez que quitamos vértices se generan nuevas caras, las cuales tendrán su propio nivel.

Con todo esto podemos ver que el paso 5 se va a repetir $O(log\, n)$ veces y cada vez que hacemos el paso 5 nos toma $O(n)$ por lo cual esa parte del algoritmo nos toma $O(n\, log\, n)$.

Y por último para construir el árbol lo podemos ir haciendo de las hojas para arriba (como en la presentación de la profesor, que se encuentra en classroom) poniendo en cada nivel las caras de la triangulación generada por la iteración del paso 5 correspondiente al nivel del árbol (solo notemos que si una cara ya está en un nivel abajo entonces no se va a subir, como en la imagen sacada de la presentación donde la cara número cuatro solo está en el último nivel y no también en el penúltimo).

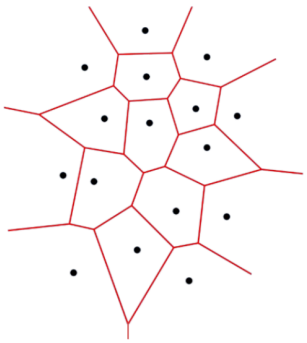


Ahora recordemos que en cada iteración del paso 5 nos va a dejar la triangulación de una subdivisión plana conexa entonces por lo visto de la complejidad geometria de esto sabemos que tendremos $O(n)$ caras, así que cada nivel a lo más tendrá

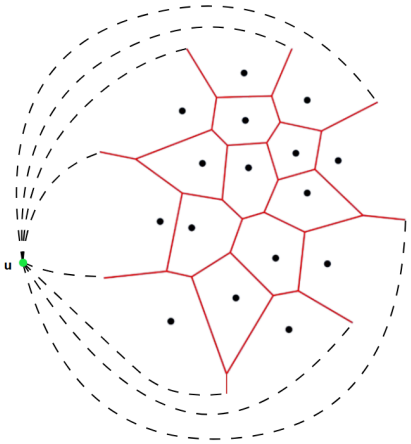
$O(n)$ caras y como el árbol tiene altura $O(log\ n)$ podemos ver que todo el árbol de los triángulos o caras lo podemos construir en $O(n\ log\ n)$.
Y de esta manera vemos que todo el proceso nos toma $O(n\ log\ n)$.
Al término de este proceso, podemos usar este árbol para buscar la cara que contienen al punto de consulta.
Para la consulta, solo tenemos que empezar en la raíz del árbol e ir bajando por el árbol hasta llegar a alguna hoja, iniciamos en la raíz revisando si el punto está dentro del triángulo grande, si esta entonces revisamos en qué triángulo o cara de los hijos de la raíz el punto está dentro y bajamos por ese hijo, así repetimos hasta llegar a una hoja del árbol y encontramos la cara que contiene a ese punto, notemos que este proceso nos toma $O(log\ n)$ ya que en cada nivel del árbol vamos a revisar si el triángulo está en $O(1)$ triángulos o caras (por lo que vimos arriba) y sabemos que ver si un punto está dentro de un triángulo podemos hacer en $O(1)$ solo checando de qué lado está el punto con respecto a las aristas del triángulo, y esto lo repetimos en cada nivel por lo tanto nos tomara lo mismo que la altura del árbol que es $O(log\ n)$.
Por último veamos la complejidad del espacio, para esto sabemos que nuestro árbol tendrá altura $O(log\ n)$ entonces solo nos falta saber cuántas caras (o nodos) tendremos en cada nivel, pero para esto ya sabemos que en cada nivel tendremos a lo más $2n - 4$ caras o $O(n)$ (por si faltan algunas constante), por lo visto la clase del 24/02/25 y que en la primera triangulación (la del paso 4) tendremos $O(n)$ entonces como en cada paso no agregamos ningún vértices y solo los quitamos entonces la cantidad de caras va disminuyendo hasta que en el primer nivel del árbol solo tengamos dos caras (la exterior y la dentro del triángulo grande), por lo tanto la cantidad de caras va disminuyendo entre más cerca de la raíz estemos, por lo tanto si cada nivel tiene a lo más $O(n)$ caras (cada cara representa un nodo) y tenemos $O(log\ n)$ altura en el árbol, entonces todo el árbol tiene una complejidad de espacio de $O(n\ log\ n)$.
En resumen tenemos que las complejidades de espacio y tiempo son las siguientes: preproceso $O(n\ log\ n)$, consulta $O(log\ n)$ y espacio $O(n\ log\ n)$.

Ejercicio 7

Sea un P un conjunto de n puntos en el plano, con $n \geq 3$, y sea V(P) el Diagrama de Voronoi de P.
Notemos que como P tiene n puntos entonces V(P) tendrá n regiones, una para cada punto.
Vamos a agregar un punto/vértice extra a V(P), al cual llamaremos u, ahora a todas las aristas que van al infinito en V(P) (es decir, las aristas que solo tiene un vértice en V(P)) las vamos a unir en el punto u, esto unir todas las aristas al infinito en el punto u (procurando que ninguna de la aristas agregadas se intersecten excepto en u).
Por ejemplo con el siguiente Diagrama de Voronoi (el visto en la exposición de la maestra).



Tendremos esto al agregar el vértice u.



Llamaremos a esta nueva subdivisión plana (esto debido a que el Diagrama de Voronoi es una subdivisión plana y como las aristas que extendemos no se intersectan excepto en u tendremos que seguirá siendo plana y es una subdivisión ya que genera regiones en el plano) como H.
Observemos que el número de caras en H, al cual llamaremos F_H, es el mismo número de caras en V(P) ($F_{V(P)}$), esto debido a que las caras que son formadas por aristas que no van al infinito en V(P) no las modificamos y todas las caras que tienen alguna arista al infinito solo las vamos a “transformar” para que solo tengan

aristas normales, por lo tanto no agregamos ninguna cara nueva y no quitamos ninguna, solo una arista que tiene aristas al infinito ahora será la cara exterior.
También el número de vértices en H, al cual llamaremos V_H, es $V_{V(P)} + 1$ con $V_{V(P)}$ siendo la cantidad de vértices en V(P), esto debido a que solo agregamos el vértice u y dejamos intactos a los demás vértices de V(P).
Y por ultimo el numero de aristas en H, al cual llamaremos E_H, es el mismo número de aristas en V(P) ($E_{V(P)}$), esto debido a que no agregamos ni quitamos ninguna arista, solo las "extendimos" para que se intersectan en el punto u.
Ahora como H es una subdivisión plana (ya que V(P) es una subdivisión plana y H sigue siendo plana y una subdivisión) entonces podemos aplicar la fórmula de Euler (vértices - aristas + caras = 2), por lo tanto tenemos que $V_H - E_H + F_H = 2$.

Otra cosa que hay que notar es que todas las aristas de H tendrá exactamente dos vértices (no tenemos aristas al infinito) y además cada vértice de H tendrá a lo más grado 3, esto debido a que en la clase del 31/03/25 vimos que un vértice de Voronoi tiene al menos grado 3 y además el punto u tendrá al menos tres aristas que inciden en él y por lo tanto al menos grado 3, ya que como $n \geq 3$ tendremos que entonces al menos tendremos 3 caras que tienen aristas al infinito las cuales van a volverse "aristas normales" al unirse en u, entonces notemos que la suma de los grados de los vértices de H es dos veces la cantidad de aristas en H (esto recordando las clases de Graficas y Juegos y que H es una gráfica) pero también notemos que como cada vértice de H tiene al menos grado 3 tendremos que la suma de los grados de los vértices de H al menos tres veces la cantidad de vértices en H, obteniendo estas igualdades (para facilitar todo esto diremos que la suma de los grados de los vértices de H lo llamaremos M_H): $M_H = 2E_H$ y $M \geq 3V_H$, por lo tanto obtenemos que $2E_H = M \geq 3V_H \Rightarrow 2E_H \geq 3V_H$.
Entonces tendremos que $2E_H \geq 3V_H$ y $V_H - E_H + F_H = 2$, por lo que podemos hacer las siguientes igualdades $V_H - E_H + F_H = 2 \Rightarrow V_H = E_H - F_H + 2$ entonces podemos sustituir de esta manera $2E_H \geq 3V_H \Rightarrow 2E_H \geq 3(E_H - F_H + 2)$, continuamos con las igualdades y tenemos que $2E_H \geq 3(E_H - F_H + 2) \Rightarrow 2E_H \geq 3E_H - 3F_H + 6 \Rightarrow 3F_H - 6 \geq E_H$, ahora recordemos lo que vimos arriba ($E_H = E_{V(P)}$ y $F_H = F_{V(P)} = n$), por lo tanto podemos sustituir $3F_H - 6 \geq E_H \Rightarrow 3n - 6 \geq E_{V(P)}$, por lo tanto las aristas de V(P) son a lo más $3n - 6$.
Ahora para los vértices usaremos que $2E_H \geq 3V_H$ y $V_H - E_H + F_H = 2$, hacemos las siguientes igualdades $V_H - E_H + F_H = 2 \Rightarrow V_H + F_H - 2 = E_H$ entonces podemos sustituir de esta manera $2E_H \geq 3V_H \Rightarrow 2(V_H + F_H - 2) \geq 3V_H$, continuamos con las

igualdades y tenemos que $2(V_H + F_H - 2) \geq 3V_H \Rightarrow 2V_H + 2F_H - 4 \geq 3V_H \Rightarrow 2F_H - 4 \geq V_H$, ahora recordemos lo que vimos arriba ($V_H = V_{V(P)} + 1$ y $F_H = F_{V(P)} = n$), por lo tanto podemos sustituir $2F_H - 4 \geq V_H \Rightarrow 2n - 4 \geq V_{V(P)} + 1$, seguimos con las igualdades y obtenemos $2n - 4 \geq V_{V(P)} + 1 \Rightarrow 2n - 5 \geq V_{V(P)}$, por lo tanto los vértices de V(P) son a lo más $2n - 5$.

Entonces podemos concluir que un Diagrama de Voronoi de un conjunto de n puntos tiene a lo más $2n - 5$ vértices y a lo más $3n - 6$ vértices.

Ejercicio 8

A. El número de vértices de $A(L)$ es a lo más $\frac{n(n-1)}{2}$

Los vértices serán las intersecciones entre las líneas.
Tendremos que cada línea puede ser interceptada por a lo más $n-1$ líneas, entonces sabemos que la cantidad de intersecciones es a lo más $\frac{n(n-1)}{2}$ (ya que estamos generando parejas entre las n líneas). Por lo cual el número de vértices de $A(L)$ es a lo más $\frac{n(n-1)}{2}$.
En el caso de que $A(L)$ sea simple tendremos que cada línea es interceptada por $n-1$ líneas esto debido a que no existen tres líneas que pasen por un punto y no existen dos líneas paralelas, entonces tendremos que todas las líneas se deben interactuar entre ellas y no en el mismo punto donde dos líneas ya se intersectan, de esta manera el número de vértices de $A(L)$ es $\frac{n(n-1)}{2}$.

B. El número de aristas de $A(L)$ es a lo más n^2

Como vimos arriba tenemos que cada línea puede ser interceptada por a lo más $n-1$ líneas, entonces tendremos que cada línea es partida en a lo más n aristas (si los vértices son las intersecciones entonces las veces en que se parte una línea, más uno, será la cantidad de aristas que genera esa línea) y como tenemos n lineas podemos concluir que número de aristas de $A(L)$ es a lo más n^2 (n aristas por cada una de las n líneas).
En el caso de que $A(L)$ sea simple tendremos que cada línea es interceptada por $n-1$ líneas (como vimos arriba), por lo cual cada línea es partida en n aristas y como tenemos n lineas podemos concluir que el número de aristas de $A(L)$ es n^2.

C. El número de caras de $A(L)$ es a lo más $\frac{n^2}{2}+\frac{n}{2}+1$

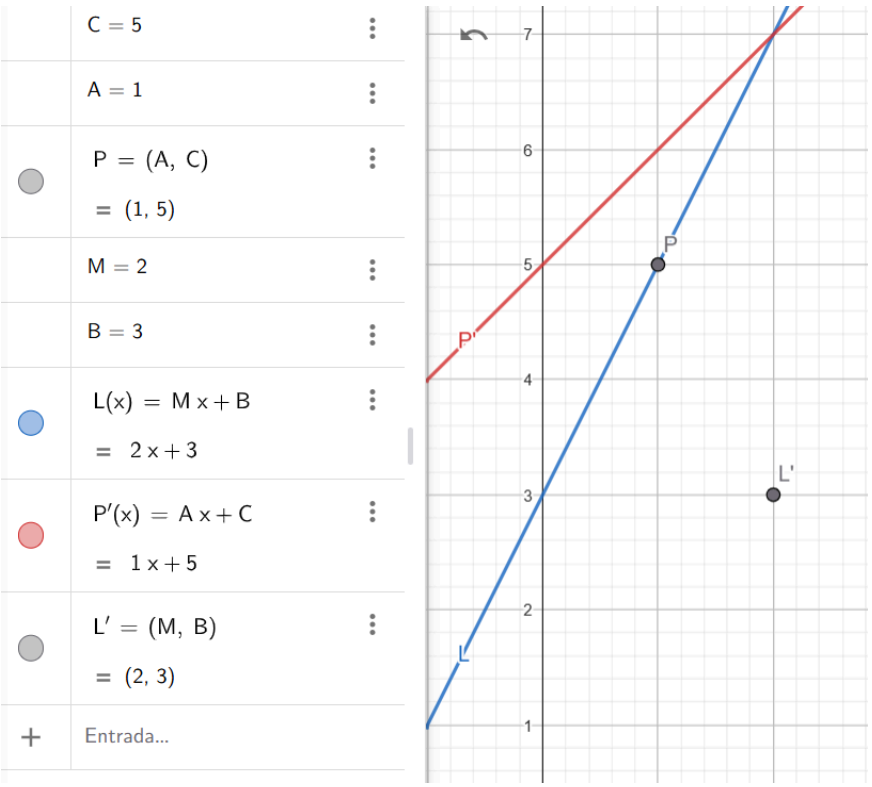
Para esto consideraremos ir agregando una línea por una línea, haciendo que al agregar una arista intersecta a todas las demás.
Al inicio no tendremos ninguna línea y tendremos una sola cara, luego agregamos una linea y ahora tendremos una línea y dos caras, después agregamos una línea que intersecta a la línea que tenemos por lo que tenemos ahora dos líneas y cuatro caras, luego agregamos otra línea que intersecta a los dos líneas que ya tenemos y así tenemos tres líneas y siete caras, así seguimos haciendo, notemos que cuando pasamos de ninguna línea a una línea las caras aumentaron en uno, cuando pasamos de una línea a dos líneas las caras aumentaron en dos, al pasar de dos a tres líneas las caras aumentaron en tres, esto pasa ya que al agregar la i-esima linea vamos a tener que se agregan i aristas (ya que la línea que intersecta a todas las demás que tenemos, recordando lo que vimos arriba) y al agregar una arista sabemos que una cara es partida en dos, por lo tanto podemos representar la

cantidad de caras con esta fórmula $1 + \sum_{i=1}^{n} i$ (ya que al inicio empezamos con una cara) y esa fórmula la podemos ver como $1 + \sum_{i=1}^{n} i = 1 + \frac{n^2}{2} + \frac{n}{2}$, entonces podemos concluir que el número de caras de $A(L)$ es a lo más $\frac{n^2}{2} + \frac{n}{2} + 1$.

En el caso de que $A(L)$ sea simple tendremos que cada línea es interceptada por $n - 1$ líneas, eso quiere decir que al agregar cada línea va a intersectar a todas las que ya había, pero sí $A(L)$ no fuera simple entonces podría pasar que una línea no intersecta a todas las demás líneas y tener que el número de caras de $A(L)$ es menor que $\frac{n^2}{2} + \frac{n}{2} + 1$, pero si $A(L)$ es simple, esto no puede pasar y todas las líneas tiene que intersectarse entre ellas y por lo tanto la explicación de agregar líneas que intersectan a todas las lineas que ya estan se debe cumplor/respectar y con eso concluimos que el número de caras de $A(L)$ es $\frac{n^2}{2} + \frac{n}{2} + 1$.

Ejercicio 9

Veamos primero si se mantiene la incidencia.
Que preserve la incidencia significa que $p \in l \Leftrightarrow l\ ^*\in p\ ^*$.
Recordemos que si $p = (p_x, p_y)$, $l = mx + b$ y $p \in l$ si y sólo si $p_y = mp_x + b$.
Entonces si se cumple $l\ ^*\in p\ ^*$ debe también de cumplirse $b = p_x m + p_y$.
Así que $b = p_x m + p_y$ y $p_y = mp_x + b$ deben cumplirse al mismo tiempo
$p_y = mp_x + b \Rightarrow b = p_y - mp_x$, tenemos $b = p_x m + p_y$ y $b = -mp_x + p_y$, luego tenemos $b = p_x m$ y $b = -mp_x$ y de ahí obtenemos $p_x m = -mp_x$, por lo tanto necesitamos que m o p_x sea 0.
Veamos un contraejemplo.
Sea $p = (1, 5)$ y sea $l = 2x + 3$, entonces veamos que se cumple $p_y = mp_x + b$ ya que $5 = (2)1 + 3 = 5$, por lo tanto se cumple que $p \in l$.
Ahora veamos que $p\ ^* = 1x + 5$ y $l\ ^* = (2, 3)$, ahora revisemos si $l\ ^*\in p\ ^*$ se cumple.
Notemos que no se cumple que $b = p_x m + p_y$ (el $p_y = mp_x + b$ que vimos arriba solo que con la transformación aplicada) ya que $b = 3$, $p_x m + p_Y = (1)2 + 5 = 7$ y $3 \neq 7$, por lo tanto no se cumple que $p \in l \Leftrightarrow l\ ^*\in p\ ^*$, así que la transformación no mantiene la incidencia.



Ahora veamos si se mantiene el orden.

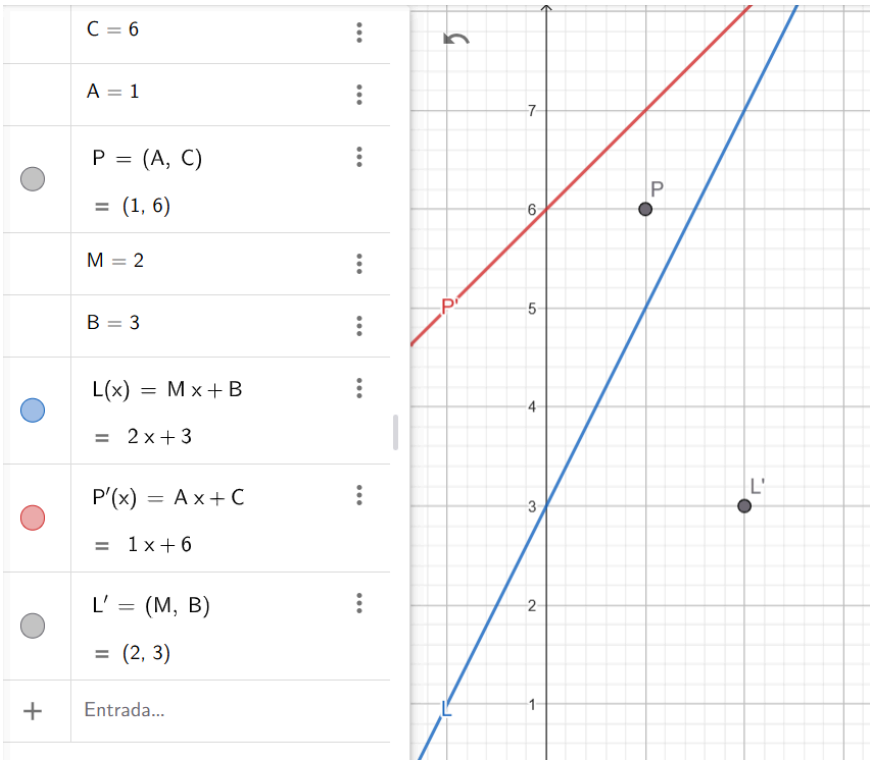
Que preserve el orden significa que si p está arriba de l (es decir $p_y > mp_x + b$) si y sólo si l * está arriba de p *, con $p = (p_x, p_y)$ y $l = mx + b$.

Veamos un contraejemplo.

Sea $p = (1, 6)$ y sea $l = 2x + 3$, entonces veamos que se cumple $p_y > mp_x + b$ ya que $6 > (2)1 + 3 = 5$, por lo tanto se cumple que p está arriba de l.

Ahora veamos que p *$= 1x + 6$ y l *$= (2, 3)$, ahora revisemos si se cumple que l * está arriba de p *.

Notemos que no se cumple que $b > p_x m + p_y$ (el $p_y > mp_x + b$ que vimos arriba solo que con la transformación aplicada) ya que $b = 3$, $p_x m + p_Y = (1)2 + 6 = 8$ y $3 < 8$, por lo tanto no se cumple que l * está arriba de p *, así que la transformación no mantiene el orden.



Por lo tanto concluimos que esta transformación no preserva la incidencia ni el orden.

Ejercicio 12

El algoritmo será el siguiente.
Lo primero que vamos a hacer es triangular el polígono simple de n vértices, esto lo podemos hacer con los algoritmos vistos en clase para triangular (el de partir en piezas monótonas y luego triangular las piezas monótonas).
Después de triangular nuestro polígonos vamos a hacer la gráfica dual de la triangulación, esto lo podemos hacer primero creando un vértice por cada triángulo generado en la triangulación (procurando relacionar cada vértice con el triángulo del que surgieron), luego de crear los vértices de las gráficas vamos a poner las aristas, tendremos que dos vértice de la gráfica están conectados por una arista si los triángulos relacionadas a esas aristas comparten una arista, esto lo vamos a hacer recorriendo las diagonales que se agregaron durante la triangulación, como sabemos que cada diagonal agregada será una arista que comparten dos triángulos de la triangulación entonces ponemos una arista entre los vértices relacionados a los triángulos que comparten la diagonal. Notemos que de esta manera vamos a relacionar cada arista de la gráfica con una diagonal agregada en la triangulación.
Lo siguiente que vamos a hacer es realizar un recorrido DFS en la gráfica dual para poder calcular la cantidad de vértices o nodos en cada subárbol, esto lo vamos a realizar de la siguiente manera, primero seleccionamos un vértice de la gráfica cualquiera y será la raíz, entonces el siguiente proceso lo empezamos desde la raíz, marcamos al vértice actual como visitado y recorremos o procesamos a todos los hijos no visitados del vértice actual, luego de recorrer a los hijos vamos a tener los tamaños de sus subárboles y para calcular el tamaño del subárbol del vértice actual vamos a tener dos casos, si el vértice actual es una hoja entonces tendremos que la cantidad de vértices o nodos en su subárbol es 1, pero si el vértice actual si tiene hijos entonces la cantidad de vértices o nodos en su subárbol es la suma de la cantidad de vértices en los subárboles de sus hijos más 1. De esta manera con un solo recorrido DFS en la gráfica dual podremos marcar a cada vértice con un número que representa la cantidad de vértices en su subárbol.
Después vamos a volver a realizar un recorrido DFS empezando por el mismo vértice o raíz donde empezó el recorrido del paso anterior, pero ahora en vez de calcular el tamaño de los subárboles vamos a ir revisando si el tamaño del subárbol del vértice que estemos procesando cumple que esté entre $n - \lfloor \frac{2n}{3} \rfloor$ y $\lfloor \frac{2n}{3} \rfloor$, es decir que $n - \lfloor \frac{2n}{3} \rfloor \leq$ el tamaño del subárbol del vértice $\leq \lfloor \frac{2n}{3} \rfloor$, cuando en el recorrido de la gráfica encontremos al vértice que satisface lo que acabamos de decir vamos a marcar a ese vértice como u y al vértice padre de u (es decir el vértice por el cual llegamos a u durante el recorrido) lo marcamos como v, entonces vamos a llamar a la arista que une los vértices u y v como a.
Por último vamos a regresar la diagonal relacionada con la arista a, es decir la diagonal agregada en la triangulación que se transformó o creo la arista a en la gráfica dual.

Esto funciona ya que la triangulación de un polígono simple de n vértices cuenta con $n-2$ triángulos y $n-3$ diagonales (esto lo vimos la clase del 14702/25), entonces como queremos encontrar una diagonal que nos divida en dos polígonos simple de a lo más $\lfloor \frac{2n}{3} \rfloor + 2$ vértices cada uno, podemos notar que si triangulamos esos polígonos tendremos que cada uno tendrá a lo más $\lfloor \frac{2n}{3} \rfloor$ triángulos, y recordemos que en la gráfica dual de la triangulación tenemos que cada vértice de la gráfica corresponde a un triángulo en la triangulación, por lo tanto si tenemos un polígono con k vértices entonces us triangulación tendrá $k-2$ triángulos y por lo tanto la gráfica dual de la triangulación tendrá $k-2$ vértices o nodos, de esta manera necesitamos partir la gráfica dual o árbol en dos parte donde cada una tenga a lo más $\lfloor \frac{2n}{3} \rfloor$ vértices, ahora notemos que si un árbol tiene n vértices entonces si lo partimos de tal forma que una parte tiene $\lfloor \frac{2n}{3} \rfloor$ vértices entonces su otra parte va a tener $n - \lfloor \frac{2n}{3} \rfloor$ vértices, y por esta razón en el segundo recorrido DFS buscamos un vértice que cumple que el tamaño de sus subárbol tenga este entre estos dos valores y de esta manera poder encontrar la diagonal que nos parta el polígono de la manera que queremos, por eso regresamos la diagonal relacionada a la arista formada entre el vértice que cumple el tamaño adecuado y su padre, para así cortar el árbol o gráfica dual y el polígono.

Por último veamos la complejidad de este algoritmo, lo primero que hacemos es triangular y esto sabemos que nos toma $O(n \log n)$ (por lo visto en las clases durante el semestre), luego para crear la gráfica dual tenemos que recorrer todos los triángulos generados en la triangulación para generar los vértices de la gráfica y luego recorremos las diagonales agregadas en la triangulación para generar las aristas de la gráfica, notemos que como la triangulación genera $n-2$ triángulos y $n-3$ diagonales entonces tendremos que generar la gráfica dual nos toma $O(n)$, después de generar la gráfica dual o árbol vamos a realizar un recorrido DFS donde vamos a calcular el tamaño de los subárboles de los vértices, notemos que para calcular el tamaño solo sumamos el tamaño de los hijos entonces calcular el tamaño lo podemos hacer en $O(1)$ (notemo que esto supone que sabemos el tamaño de los subárboles de los hijos pero esto lo tendremos por el recorrido ya que calculamos los tamaños de los hijos antes que el del padre) y como hacemos un recorrido DFS (en generar un recorrido sobre el árbol o gráfica) entonces nos va tomar $O(n)$ en total, esto debido a que calcular el tamaño es $O(1)$ y el recorrido nos toma $O(V+E)$ con V la cantidad de vértice y E la cantidad de aristas pero como sabemos que cada triángulo de la triangulación representa un vértice en la gráfica y cada diagonal de la triangulación representa una arista en la gráfica tendremos que $V = n-2$ y $E = n-3$, por lo tanto el recorrido nos toma $O(n)$, posteriormente, realizamos otro recorrido DFS pero ahora para revisar que vértice cumple que su tamaño este entre $n - \lfloor \frac{2n}{3} \rfloor$ y $\lfloor \frac{2n}{3} \rfloor$, para esto solo revisamos el tamaño de los subárboles de los vértices y lo podemos hacer en $O(1)$ y como también hacemos un recorrido

entonces todo el recorrido y encontrar al vértice que lo cumple nos toma $O(n)$, y por último regresar la diagonal relacionada a la arista entre el vértice encontrado y su padre lo podemos hacer en a lo más $O(n)$, esto ya que si hacemos otro recorrido para encontrar al padre (si es que no guardamos el padre de los vértices durante el recorrido) nos toma $O(n)$ y luego recorremos las diagonales para ver cual de estas está relacionada a la arista que encontramos.
En total todo el algoritmo nos va a tomar $O(n\ log\ n)$.