Cubiertas convexas II

Dr. Eduardo A. Rodríguez Tello

Cinvestav-Tamaulipas

22 de enero del 2013

1. Cubiertas convexas II
   - Algoritmo Divide y Vencerás
   - Algoritmo QuickHull
   - Algoritmo de Caminata de Jarvis
   - Tarea

Algoritmo Divide y Vencerás

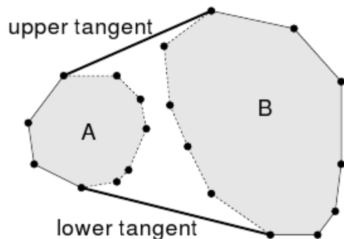
- El problema de construcción de la cubierta convexa para un conjunto P de puntos en el plano pueden ser resuelto utilizando diferentes enfoques
- A continuación estudiaremos otro algoritmo de orden $O(n \log n)$ el cual está basado en la técnica de diseño conocida como **Divide y Vencerás**
- Puede ser vista como una generalización del famoso algoritmo de ordenamiento *MergeSort*

Algoritmo Divide y Vencerás

Divide y Vencerás

1. Ordenar los puntos en P de acuerdo a su coordenada x
2. Particiona el conjunto de puntos P en dos conjuntos A y B, donde A consiste de los $\lceil n/2 \rceil$ puntos con las coordenadas x más pequeñas (izq.) y B los $\lfloor n/2 \rfloor$ puntos restantes (der.)
3. Calcula recursivamente $H_A = conv(A)$ y $H_B = conv(B)$
4. Combina las dos cubiertas en una cubierta convexa, H, mediante el cálculo de las tangentes superior e inferior de H_A y H_B y descartando todos los puntos que caen entre esas dos tangentes

Algoritmo Divide y Vencerás



Algoritmo Divide y Vencerás

- El ordenamiento del primer paso garantiza que los conjuntos A y B estén separados por una línea vertical, lo cual asegura que A y B no se traslapan
- Esto simplifica el paso de combinación (paso 4)
- Los pasos 2, 3 y 4 se repiten en cada nivel de recursión, deteniéndose cuando $n \leq 3$
- Si $n = 3$ la cubierta es un triángulo (asumiendo que no hay 3 puntos colineales)

Algoritmo Divide y Vencerás

- El tiempo de ejecución asintótico del algoritmo puede expresarse mediante una relación de recurrencia
- Dada una entrada de tamaño n, consideraremos el tiempo necesario para realizar todos los pasos del algoritmo, ignorando las llamadas recursivas
- Esto incluye el tiempo para:
  1. Particionar el conjunto de puntos,
  2. Calcular las dos tangentes, y
  3. Regresar el resultado final
- Claramente los pasos 2 y 3 pueden ser realizados en tiempo $O(n)$, asumiendo que los vértices de la cubierta son representados con una lista ligada

Algoritmo Divide y Vencerás

- A continuación veremos como las tangentes pueden ser calculadas en tiempo $O(n)$
- Por lo tanto, ignorando los factores constantes, podemos describir el tiempo de ejecución mediante la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & \text{si} \quad n \leq 3 \\ n + 2T(n/2) & \text{sino} \end{cases} \tag{1}$$

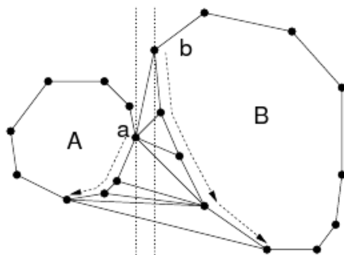
Algoritmo Divide y Vencerás

- Esta es la misma relación de recurrencia que surge en el algoritmo *MergeSort*
- Es fácil mostrar que $T(n) \in O(n \log n)$
- Solamente falta mostrar cómo calcular las dos tangentes
- Algo que simplifica el proceso de calcular las tangentes es que los dos conjuntos A y B están separados por una línea vertical
- Esto asumiendo que los puntos no tienen coordenadas x duplicadas

Algoritmo Divide y Vencerás

- Vamos a concentrarnos en la tangente inferior (la superior es simétrica)
- El algoritmo funciona mediante un procedimiento simple de “caminata”
- Inicializamos a como el punto más a la derecha de H_A y b el más a la izquierda de H_B (esto puede ser encontrado en tiempo lineal)

Algoritmo Divide y Vencerás



Algoritmo Divide y Vencerás

- La tangencia inferior es una condición que puede ser verificada localmente mediante una prueba de orientación involucrando los dos vértices y vértices vecinos en la cubierta
- Se iteran los siguientes dos ciclos, los cuales avanzan a y b hasta que ellos alcanzan los puntos de tangencia inferior

Algoritmo Divide y Vencerás

tangenteInferior(H_A, H_B)

1. Sea a el punto más a la derecha de H_A
2. Sea b el punto más a la izquierda de H_B
3. **While** (ab no es una tangente inferior de H_A y H_B) **do**
   1. **While** (ab no es una tangente inferior de H_A) **do** $a \leftarrow a.pred$ (mover a ↻)
   2. **While** (ab no es una tangente inferior de H_B) **do** $b \leftarrow b.succ$ (mover b ↺)
4. Regresar ab

Algoritmo Divide y Vencerás

- Dado un punto en la cubierta, sea $a.succ$ y $a.pred$ su sucesor y predecesor en orden $\circlearrowleft$ con respecto a la cubierta
- La condición *"ab no es una tangente inferior de H_A"* puede ser implementada con la prueba de orientación $Orient(b, a, a.pred) \leq 0$
- Esta prueba es similar con H_B

Algoritmo Divide y Vencerás

- Probar la correctez de este procedimiento es algo no tan complicado, el secreto está en probar que los dos while internos nunca van más allá de los puntos de la tangente inferior
- La prueba detallada puede ser consultada en el libro de O'Rourke
- El punto importante es que cada vértice en la cubierta puede ser visitado máximo una vez por cada búsqueda, y por lo tanto su tiempo es $O(m)$, donde $m = |H_A| + |H_B| = |A| + |B|$
- Esto es exactamente lo que se requiere para tener un tiempo de ejecución $O(n \log n)$

1 Cubiertas convexas II
- Algoritmo Divide y Vencerás
- Algoritmo QuickHull
- Algoritmo de Caminata de Jarvis
- Tarea

Algoritmo QuickHull

- Si el algoritmo Divide y Vencerás puede ser visto como una generalización del *MergeSort*, podríamos preguntarnos si existen las generalizaciones correspondientes a otros algoritmos de ordenamiento para calcular cubiertas convexas
- En particular, el siguiente algoritmo que vamos a considerar puede ser visto como la generalización del *QuickSort*
- El algoritmo resultante es conocido como **QuickHull**

Algoritmo QuickHull

- Al igual que el *QuickSort* corre en tiempo $O(n \log n)$ para entradas favorables pero puede tomar tiempo $O(n^2)$ con datos desfavorables
- Sin embargo, a diferencia del *QuickSort*, no hay una forma obvia de convertirlo en un algoritmo aleatorizado con tiempo de ejecución esperado $O(n \log n)$
- No obstante, **QuickHull** tiende a desempeñarse muy bien en la práctica

Algoritmo QuickHull

- La intuición indica que en muchas aplicaciones la mayoría de los puntos caen en el interior de la cubierta
- Por ejemplo, si los puntos están uniformemente distribuídos en un cuadrado unitario, entonces puede ser mostrado que el número esperado de puntos en la cubierta convexa es $O(\log n)$

Algoritmo QuickHull

- La idea detrás del algoritmo *QuickHull* es descartar los puntos que no están en la cubierta tan pronto como sea posible
- *QuickHull* comienza por calcular dos puntos extremos
- Utilizaremos el punto x más a la derecha y más abajo y el punto y más a la izquierda y más arriba
- Claramente estos puntos deben estar en la cubierta

Algoritmo QuickHull

- La cubierta convexa completa estará formada por una cubierta superior sobre xy y una cubierta inferior bajo xy
- *QuickHull* encuentra estas cubiertas mediante un procedimiento que inicia con los puntos extremos (a, b)
- Continua encontrando un tercer punto extremo c estrictamente a la derecha de ab
- Elimina todos los puntos dentro del triángulo abc
- Itera recursivamente en (a, c) y (c, b)

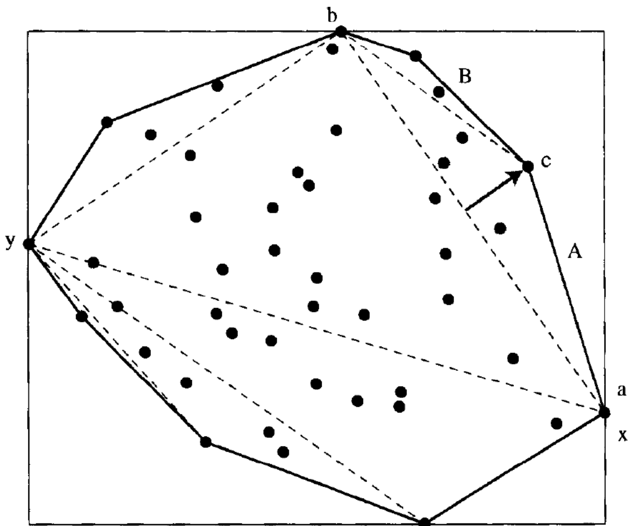
Algoritmo QuickHull

- Sea S el conjunto de puntos estrictamente a la derecha de ab (S podría estar vacío)
- La idea clave es que el punto $c \in S$ que está más alejado de ab debe estar en la cubierta convexa
- De hecho es un punto extremo en la dirección ortogonal a ab
- Por lo tanto se pueden eliminar todos los puntos sobre o dentro del triángulo abc (excepto por a, b y c)

Algoritmo QuickHull

- El resto de los puntos es dividido en dos subconjuntos A y B:
  - A, contiene aquellos puntos que caen fuera (a la derecha) de ac
  - B, incluye aquellos puntos que caen fuera (a la derecha) de cb
- Podemos clasificar cada punto p, calculando las orientaciones de las tripletas acp y cbp
- Se reemplaza la arista ab con ac y cb para continuar recursivamente el procedimiento

Algoritmo QuickHull



Algoritmo QuickHull

QuickHull(a, b, S)

1. **if** $S = \emptyset$ **then return** $\emptyset$
2. **else if** $S = a, b$ **then return** (a, b) (una arista de la cubierta)
3. **else**
   - $c \leftarrow$ índice del punto con máxima distancia de ab
   - $A \leftarrow$ conjunto de puntos estrictamente a la derecha de (a, b)
   - $B \leftarrow$ conjunto de puntos estrictamente a la derecha de (c, b)
   - **return** QuickHull$(a, c, A) \cup (c) \cup$ QuickHull(c, b, B)

Algoritmo QuickHull

- Ahora analizaremos de la complejidad temporal del algoritmo *QuickHull*
- Encontrar los puntos extremos x y y, y particionar S en A y B puede ser realizado en tiempo $O(n)$
- En cuanto a la función recursiva, supongamos que $|S| = n$, entonces toma n pasos determinar el punto extremo c
- Pero el costo de las llamadas recursivas depende del tamaño de los conjuntos A y B

Algoritmo QuickHull

- Sea $|A| = \alpha$ y $|B| = \beta$ con $\alpha + \beta \leq n - 1 = O(n)$
- La suma es como máximo $n - 1$ porque c no está incluido ni en A ni en B
- Si la complejidad temporal de llamar *QuickHull* con $|S| = n$ es $T(n)$, podemos expresar T recursivamente en términos de ella misma

$$T(n) = O(n) + T(\alpha) + T(\beta) \tag{2}$$

- Para resolver esta ecuación es necesario expresar α y β en términos de n

Algoritmo QuickHull

- Considerando el mejor caso posible, i.e. cuando cada división está lo más balanceada posible: $\alpha = \beta = n/2$
- Por lo tanto se tiene que

$$T(n) = O(n) + 2T(n/2) \tag{3}$$

- Esta relación de recurrencia, nos es familiar, y su solución es: $T(n) = O(n \log n)$
- Por lo tanto el mejor tiempo de ejecución para el algoritmo *QuickHull* es $O(n \log n)$ (cuando los puntos están aleatoriamente distribuidos)

Algoritmo QuickHull

- Por otra parte, el peor caso para este algoritmo ocurre cuando la división está demasiado desbalanceada: $\alpha = 0$ y $\beta = n - 1$
- Se llega a la siguiente relación de recurrencia

$$T(n) = O(n) + T(n-1) = cn + T(n-1) \tag{4}$$

- La expansión repetida de esta relación de recurrencia muestra que $T(n) = O(n^2)$
- Por lo tanto a pesar de que el algoritmo *QuickHull* es generalmente muy rápido, es cuadrático en el peor caso a diferencia del algoritmo de *Graham* cuyo peor caso es $O(n \log n)$

1. Cubiertas convexas II
   - Algoritmo Divide y Vencerás
   - Algoritmo QuickHull
   - Algoritmo de Caminata de Jarvis
   - Tarea

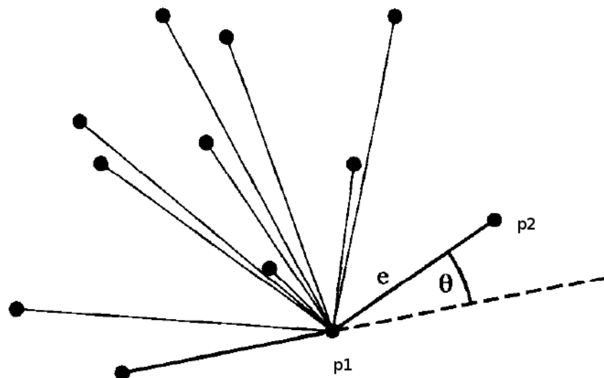
Algoritmo de Caminata de Jarvis

- El algoritmo de **Caminata de Jarvis** también es conocido como el método de *envoltura de regalo*
- Dado un conjunto S de n puntos en el plano, supongamos que movemos una línea L recta barriendo el plano hasta que L haga contacto con un punto p_1 de S
- El punto p_1 debe estar en la frontera de la cubierta convexa de S dado que hasta ese momento, todos los puntos de S están situados a un lado de la línea L y p_1 sobre la línea

Algoritmo de Caminata de Jarvis

- A continuación la línea L es girada sobre el punto p_1, en sentido contrario a las manecillas del reloj, hasta que L haga contacto con otro punto p_2 de S
- El segmento $\overline{p_1 p_2}$ es entonces una arista de la cubierta convexa dado que nuevamente todos los puntos de S están situados a un lado de la línea L y el segmento $\overline{p_1 p_2}$ está sobre la línea L
- Ahora L es girada sobre p_2, en sentido contrario a las manecillas del reloj, hasta que L toque un tercer punto p_3 de S
- El segmento $\overline{p_2 p_3}$ es la segunda arista de la cubierta convexa
- El proceso continua hasta llegar nuevamente al punto p_1 y de esta forma cerrar la cubierta convexa

Algoritmo de Caminata de Jarvis



Algoritmo de Caminata de Jarvis

- Este proceso puede ser visto como un método de envoltura.
- Supongamos que fijamos el extremo de una cuerda en el punto p_1 que se sabe es un vértice de la cubierta
- Entonces intentamos envolver los puntos con la cuerda
- La cuerda obviamente representa la frontera de la cubierta convexa

Algoritmo de Caminata de Jarvis

- Estudiemos a detalle el proceso anterior
- Supongamos un punto intermedio de la ejecución del algoritmo en el cual se han encontrado los vértices de la cubierta convexa $p_1, p_2, \cdots, p_i$
- ¿Qué punto será el próximo vértice de la cubierta?
- Obviamente, será el primer punto p_{i+1} tocado por la línea, cuando es girada sobre el punto p_i
- El ángulo $\angle p_{i-1} p_i p_{i+1}$ debe ser el más grande

Algoritmo de Caminata de Jarvis

CaminataDeJarvis(S)

1. $H = \emptyset$ (lista de vértices de la cubierta convexa)
2. $p_1 \leftarrow$ el punto en S que tiene la coordenada y más pequeña
3. $p_2 \leftarrow$ el punto en S tal que la pendiente de $\overline{p_1 - p_2}$ es la más pequeña, con respecto al eje x
4. Agregar p_1 y p_2 a H
5. $i \leftarrow 2$
6. **while** $p_i \neq p_1$ **do**
   - $p_{i+1} \leftarrow$ el punto en S tal que el $\angle p_{i-1} p_i p_{i+1}$ es el más grande
   - $i \rightarrow i + 1$‘
   - Agregar p_i a H
7. **return** H

Algoritmo de Caminata de Jarvis

- Estudiemos ahora la complejidad de este algoritmo
- Suponga que hay k vértices en la cubierta convexa H de un conjunto de puntos S
- Los puntos p_1 y p_2 son obviamente vértices de H
- Además, es claro que encontrar los puntos p_1 y p_2 toma un tiempo $O(n)$, asumiendo que $|S| = n$

Algoritmo de Caminata de Jarvis

- Para encontrar el siguiente vértice de la cubierta p_{i+1}, necesitamos verificar el ángulo $\angle p_{i-1} p_i p_{i+1}$ para cada punto p en S
- Este paso toma un tiempo $O(n)$ en cada vértice de la cubierta
- Por lo tanto el algoritmo de *Caminata de Jarvis* corren en tiempo $O(kn)$
- Notemos que si k es asintóticamente más pequeño que $\log n$ ($o(\log n)$) entonces este algoritmo es mejor que el de Graham, ya que correría en tiempo lineal
- Sin embargo, si k es más grande, de manera que $k = \Omega(n)$, entonces la complejidad temporal del algoritmo de *Caminata de Jarvis* es $\Omega(n^2)$

1. Cubiertas convexas II
   - Algoritmo Divide y Vencerás
   - Algoritmo QuickHull
   - Algoritmo de Caminata de Jarvis
   - Tarea

Tarea

- Fecha de entrega: 5 de febrero del 2013 antes de las 12h00
- Implementar los algoritmos siguientes para calcular la cubierta convexa de un conjunto de puntos:
  - *Divide y Vencerás*
  - *QuickHull*
  - *Caminata de Jarvis*
- Realizar un reporte donde se efectúe un análisis de los tres algoritmos en cuanto a su desempeño con respecto al escalamiento del tamaño de las instancias de prueba generadas aleatoriamente

Tarea ...

- Construya un conjunto W de casos de prueba que fuercen *QuickHull* a su peor caso ($O(n^2)$) y grafique el desempeño del algoritmo con respecto al caso promedio con instancias del mismo tamaño
- A la gráfica resultante del punto anterior agregue la curva del desempeño del algoritmo de Graham para el mismo conjunto W de casos de prueba
- Comente en el reporte las conclusiones a que llega con estas gráficas