Algoritmos de fuerza bruta

Dr. Eduardo A. Rodríguez Tello

Cinvestav-Tamaulipas

29 de enero de 2018

1. Algoritmos de fuerza bruta
   - Introducción
   - Ordenamiento por selección
   - Búsqueda de subcadenas *(string matching)*
   - Intersección de segmentos de línea
   - Cubierta convexa
   - Fortalezas y debilidades

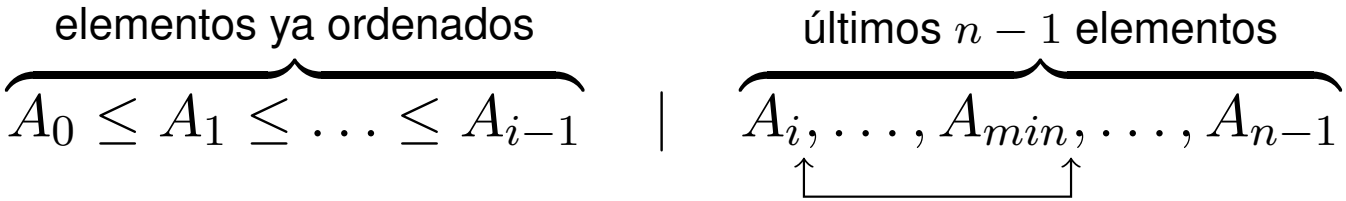
Introducción

- Es un enfoque de solución, usualmente basado en el enunciado del problema y las definiciones de conceptos involucrados
- Ejemplos:
  - Calcular a^n ($a > 0$, $n \in \mathbb{Z}^+$)
  - Calcular $n!$
  - Multiplicar dos matrices
  - Buscar un valor dentro de una lista

Ordenamiento por selección

ALGORITHM *SelectionSort*$(A[0..n-1])$

//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$



Ordenamiento por selección

Analicemos la eficiencia del algoritmo de ordenamiento por selección

- El tamaño de la entrada es el tamaño del arreglo n
- La operación básica es la comparación $A[j] < A[min]$
- El número de veces que ésta es ejecutada depende sólo de n y puede ser expresada con la siguiente sumatoria:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1)-(i+1)+1] = \sum_{i=0}^{n-2} (n-1-i) = \frac{n(n-1)}{2}$$

- Por lo tanto el ordenamiento por selección es un algoritmo $\Theta(n^2)$

Búsqueda de subcadenas *(string matching)*

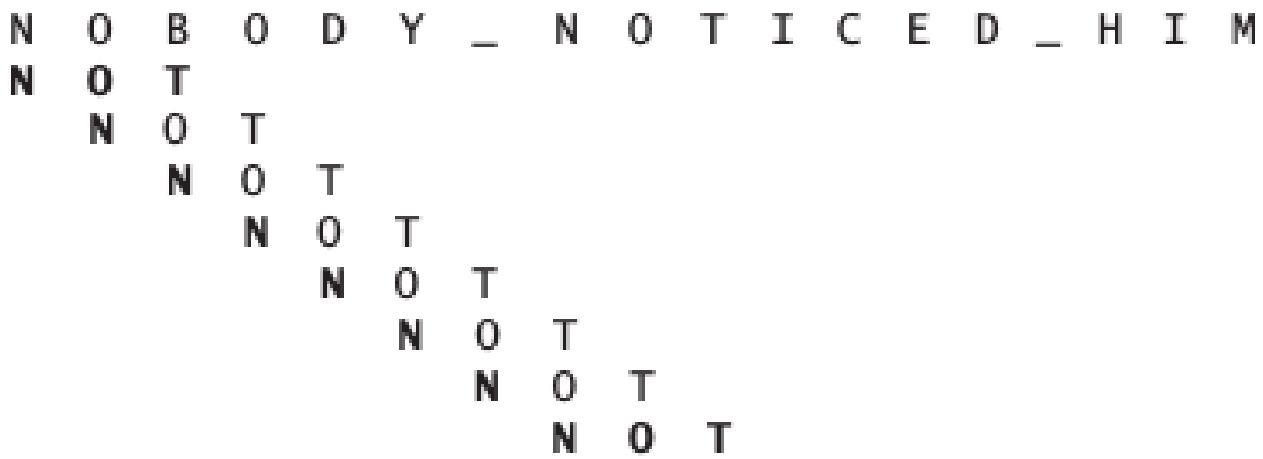
Problema de búsqueda de subcadenas (*string matching*)

Dada una cadena de n caracteres, llamada *texto*, y una cadena de m caracteres ($m < n$), llamada *patrón*, encontrar una subcadena del texto que coincida con el patrón. De manera más formal, se requiere encontrar el índice i del carácter más a la izquierda de la primera subcadena que coincida en el texto, tal que:

$$t_i = p_0, \ldots, t_{i+j} = p_j, \ldots, t_{i+m-1} = p_{m-1}$$

Si se requieren todas las subcadenas coincidentes entonces se pude seguir analizando todo el texto.

Búsqueda de subcadenas *(string matching)*



Búsqueda de subcadenas *(string matching)*

ALGORITHM *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

- ¿Cuál es el peor caso?

Búsqueda de subcadenas *(string matching)*

ALGORITHM *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

- ¿Cuál es el peor caso?
- El algoritmo puede tener que hacer m comparaciones antes de avanzar el patrón, y esto para cada uno de los $n - m + 1$ intentos
- $m(n - m + 1) \in \Theta(nm)$

Intersección de segmentos de línea

- Uno de los problemas más básicos en geometría computacional es el del cálculo de intersecciones
- El cálculo de intersecciones en espacios 2 y 3 dimensionales es esencial para diversas áreas de aplicación
- En **modelado de sólidos** las personas suelen crear formas complejas mediante la aplicación de diversas operaciones booleanas (intersección, unión, y diferencia) a simples formas primitivas. El proceso es llamado *geometría constructiva de sólidos* (CSG). Con el fin de realizar estas operaciones, el paso más básico es determinar los puntos en los que los límites de dos objetos se intersectan

Intersección de segmentos de línea

- En **robótica** y **planificación del movimiento** es importante saber cuando dos objetos se intersectan para detectar y evitar colisiones
- En los **sistemas de información geográfica** es útil la superposición de dos subdivisiones (e.g., una red de carreteras y los límites de un estado para determinar las responsabilidades de mantenimiento de carreteras). Dado que estas redes se forman a partir de colecciones de segmentos de línea, esto genera el problema de la determinación de las intersecciones de segmentos de línea

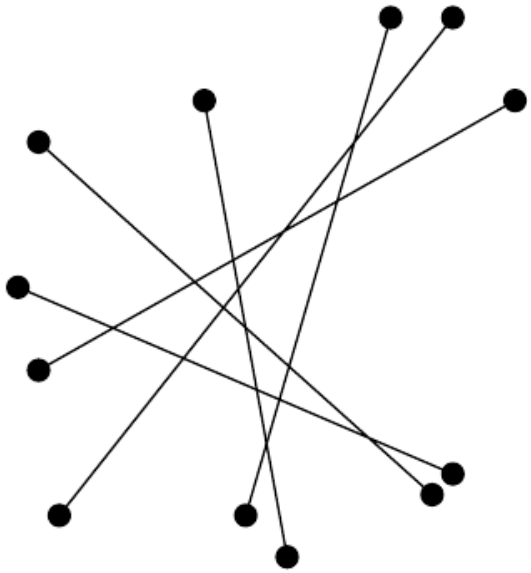
Intersección de segmentos de línea

- En **gráficas computacionales**, *ray shooting* es un importante método para renderizar escenas. Computacionalmente la parte más intensiva de este método es determinar la intersección de los rayos con otros objetos.
- La mayoría de los problemas de intersección complejos se descomponen sucesivamente en subproblemas más simples como el de intersección de segmentos de línea que veremos a continuación.

Intersección de segmentos de línea

Problema de intersección de segmentos de línea

Dado un conjunto de n segmentos de línea en el plano, cada uno representado con las coordenadas de sus dos puntos extremos, encontrar todos los puntos donde un par de segmentos de línea se intersectan



Cinvestav

Intersección de segmentos de línea

- ¿Cuántos posibles puntos de intersección pueden existir?

Intersección de segmentos de línea

- ¿Cuántos posibles puntos de intersección pueden existir?
- Observemos que n segmentos de línea pueden tener un número total de puntos de intersección que va desde 0 hasta $\binom{n}{2}$

Intersección de segmentos de línea

Algoritmo SlowIntersectionPoints

Input: Un conjunto L de n segmentos de línea en el plano
Output: Un conjunto I de puntos de intesección encontrados

1 $I \leftarrow \emptyset$ **for** *cada par ordenado* a, b *de segmentos de línea en* $L \times L$, *donde* $a \neq b$ **do**
2 $\quad I \leftarrow$ *VerificarIntersección*(a, b)
3 **end**

Intersección de segmentos de línea

Algoritmo SlowIntersectionPoints

Input: Un conjunto L de n segmentos de línea en el plano
Output: Un conjunto I de puntos de intesección encontrados

1 $I \leftarrow \emptyset$ **for** *cada par ordenado* a, b *de segmentos de línea en* $L \times L$, *donde* $a \neq b$ **do**
2 $\quad I \leftarrow$ *VerificarIntersección*(a, b)
3 **end**

- ¿Qué complejidad tiene este algoritmo?

Intersección de segmentos de línea

Algoritmo SlowIntersectionPoints

Input: Un conjunto L de n segmentos de línea en el plano
Output: Un conjunto I de puntos de intesección encontrados

1 $I \leftarrow \emptyset$ **for** *cada par ordenado* a, b *de segmentos de línea en* $L \times L$, *donde* $a \neq b$ **do**
2 $\quad I \leftarrow$ *VerificarIntersección*(a, b)
3 **end**

- ¿Qué complejidad tiene este algoritmo?
- $O(n^2)$ por que el número total de puntos de intersección puede ser hasta $\binom{n}{2} = \frac{n!}{2!(n-2)!}$

Cubierta convexa

Problema (planar) de la cubierta convexa

- Dado un conjunto P de n puntos en el plano, calcular la representación del polígono convexo cerrado que representa la cubierta convexa de P
- La representación más simple de una cubierta convexa es la enumeración en el sentido de las manecillas del reloj ($\circlearrowright$) de sus vértices
- Idealmente la cubierta convexa debe consistir sólo de los puntos extremos, en el sentido que si tres puntos caen en un vértice de la frontera de la cubierta convexa, entonces el punto medio no debe ser tomado en cuenta como parte de la cubierta

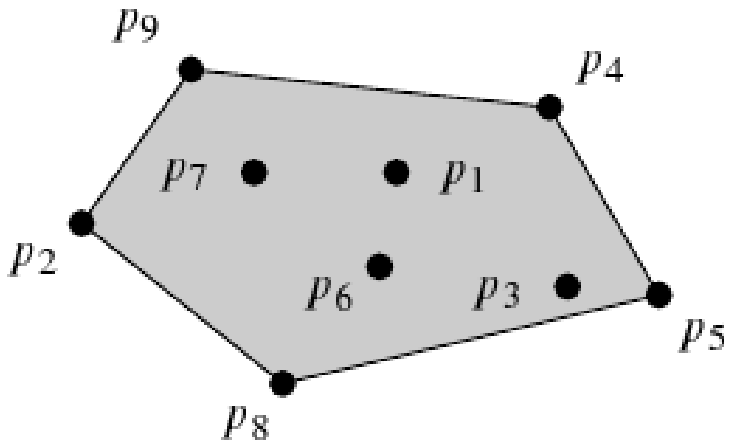
Cubierta convexa

input = set of points:

$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

output = representation of the convex hull:

p_4, p_5, p_8, p_2, p_9



- El problema puede ser abordado más fácilmente si asumimos que los puntos están en la posición general, y en particular que no hay tres colineales

Cubierta convexa, Ejercicio

- Plantee el pseudocódigo de un algoritmo de fuerza bruta para resolver este problema
- Determine la complejidad temporal del algoritmo

Cubierta convexa

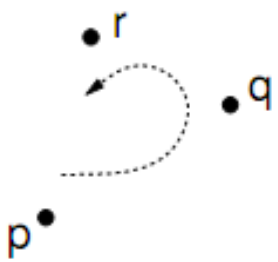
Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
1.  E ← ∅.
2.  for all ordered pairs (p, q) ∈ P × P with p not equal to q
3.      do valid ← true
4.          for all points r ∈ P not equal to p or q
5.              do if r lies to the left of the directed line from p to q
6.                  then valid ← false.
7.          if valid then Add the directed edge pq→ to E.
8.  From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise order.
```

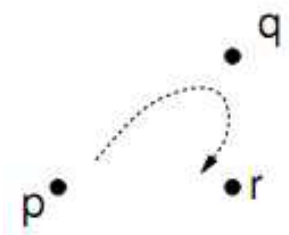
- Se verifican $n^2 - n$ pares de puntos. Cada par se compara con otros $n - 2$ puntos, lo que toma $O(n^3)$. El paso final toma $O(n^2)$
- El tiempo total de ejecución es $O(n^3)$, ¿Podrá hacerse más eficientemente?

Cubierta convexa

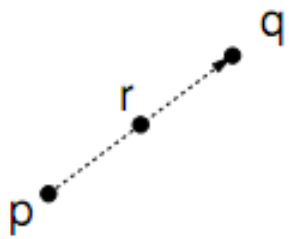
$Orient(p, q, r) > 0$



$Orient(p, q, r) < 0$



$Orient(p, q, r) = 0$



Fortalezas y debilidades

Fortalezas

- Amplia aplicabilidad
- Simplicidad
- Lleva a diseñar algoritmos que permiten resolver razonablemente algunos problemas importantes (e.g., multiplicación de matrices, ordenamiento, búsqueda de subcadenas, etc.)
- Pueden ser punto de referencia para desarrollar algoritmos más efectivos

Debilidades

- Raramente llevan a desarrollar algoritmos altamente eficientes
- Algunos algoritmos de fuerza bruta son extremadamente lentos (inaceptables)