

Tarea 6 - Programación y algoritmos 2

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En la presente tarea se describirá el código implementado para resolver el problema del cálculo de la fuerza de los individuos en su versión en 2 dimensiones. De la misma forma se hablará sobre el código implementado, tanto en su versión de fuerza bruta y con el paradigma *Divide y Vencerás*, finalmente se hará una comparación y se hablará de los resultados obtenidos.

1. Descripción del problema

Dado un conjunto de puntos $\{x_i, y_i\}_{i=1}^n$ se quiere buscar para cada punto la fuerza del mismo, es decir, a cuántos puntos este domina. El concepto de dominancia es el siguiente: Dados dos puntos $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$ decimos que P_2 domina a P_1 si: $x_2 \geq x_1$ y $y_2 \geq y_1$.

2. Implementación con Fuerza Bruta

Uno de los algoritmos que resuelve el problema anterior es el basado en la técnica de fuerza bruta, es decir, para cada punto P_i podemos ver a cuántos puntos este domina haciendo un recorrido de puntos $P_j, \forall j \in [1, n] \wedge j \neq i$. La complejidad del algoritmo anterior sería $O(n^2)$ ya que por cada punto hay que hacer un recorrido por todo nuestro conjunto. Podemos ver que si el número de puntos crece, la eficiencia del algoritmo es mala. A continuación se muestra el pseudocódigo de dicho algoritmo.

Algorithm 1 Dominancia de Puntos Fuerza Bruta.

```
1: procedure GETPOINTSSTRENGTHNAIVE(Points)
2:    $n \leftarrow \text{Points.size}()$ 
3:    $\text{result} \leftarrow \{\}$ 
4:   for  $i = 0 \rightarrow n - 1$  do
5:     for  $j = 0 \rightarrow n - 1$  do
6:       if  $i = j$  then
7:         continue
8:        $\text{cont} \leftarrow 0$ 
9:       if  $\text{Points}[i].x \geq \text{Points}[j].x$  and  $\text{Points}[i].y \geq \text{Points}[j].y$  then
10:         $\text{cont} \leftarrow \text{cont} + 1$ 
11:        $\text{result} \leftarrow \text{result} \cup \{\text{cont}\}$ 
12:   return  $\text{result}$ 
```

3. Implementación con Divide y Vencerás

Un mejor algoritmo se hace basado en la famosa técnica *Divide y Vencerás*, la cual consiste en dividir el problema en subproblemas más pequeños, resolver estos de una forma eficiente y finalmente mezclar las soluciones para generar la solución del problema original.

El algoritmo para encontrar la fuerza del conjunto de puntos consiste en:

1. Ordenar el conjunto de puntos por su componente X .
2. Dividir el conjunto de puntos en dos subconjuntos de igual tamaño $(\frac{n}{2})$.
3. Obtener las fuerzas de ambos conjuntos recursivamente.
4. Mezclar las soluciones para encontrar las fuerzas del conjunto original.

Del algoritmo anterior hay que complementar con el caso base del mismo, el cual consiste en un conjunto donde sólo existe un punto, y para ese caso la fuerza del mismo sería 0 ya que no hay otros puntos para dominar.

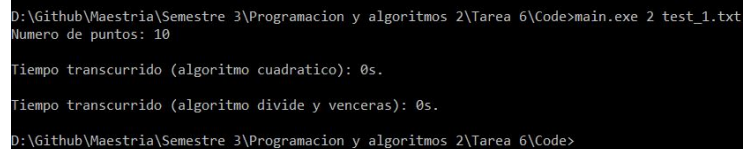
Además la mezcla de las soluciones no es algo trivial, la misma consiste en darse cuenta que las fuerzas de los puntos del conjunto del lado izquierdo ya están correctamente calculadas, más aún las del lado derecho no, y para actualizarlas hay que fijarse en la componente Y de los puntos ya que esta es la única que afecta en el nuevo cálculo. Para mezclar se ordenan los dos conjuntos en base a la componente Y y se aplica una técnica parecida a la función *merge* del *mergesort* en donde se tienen dos índices que apuntan a los siguientes elementos más pequeños de los conjuntos, además se lleva un contador que inicialmente es igual a 0. Comparamos los elementos por su coordenada Y , si el elemento de la izquierda es mayor al de la derecha, entonces agregamos el elemento de la derecha a la nueva lista pero sumando a su fuerza el valor que tiene el contador, en caso contrario que el elemento de la derecha sea mayor, sumamos 1 al contador (ya que vimos que el de la derecha domina al de la izquierda) y agregamos el elemento de la izquierda a la nueva lista. De la misma forma hay que cuidar los casos donde se terminan los elementos de un lado o de otro. De esta forma la complejidad del algoritmo anterior es $O(n \log^2 n)$ ya que en cada mezcla hay que ordenar ambos subconjuntos en $O(m \log m)$. Para mejorar la técnica anterior y evitar hacer un método de ordenación normal se puede generar otro vector de puntos ordenado por Y y usarlo como referencia para ordenar los subconjuntos en $O(m)$, haciendo esto la complejidad de todo el algoritmo baja a $O(n \log n)$. A continuación se muestra el pseudocódigo del algoritmo anterior descrito.

Algorithm 2 Dominancia de Puntos Divide y Vencerás.

```
1: procedure GETPOINTSSTRENGTHDAC(SortedXPoints, SortedYPoints, lo, hi)
2:    $n \leftarrow hi - lo + 1$ 
3:    $result \leftarrow \{\}$ 
4:   if  $n = 1$  then
5:      $result[0] \leftarrow SortedXPoints[lo].p \cup 0$   $\triangleright$  Se regresa el punto con fuerza 0
6:     return  $result$ 
7:    $mid \leftarrow (lo + hi) / 2$ 
8:    $force1 \leftarrow GetPointsStrengthDaC(SortedXPoints, SortedYPoints, lo, mid)$ 
9:    $force2 \leftarrow GetPointsStrengthDaC(SortedXPoints, SortedYPoints, mid + 1, hi)$ 
 $\triangleright$  Se asume que los subconjuntos ya vienen ordenados por la componente Y
10:   $i \leftarrow 0$ 
11:   $j \leftarrow 0$ 
12:   $cont \leftarrow 0$ 
13:  for  $k = 0 \rightarrow n - 1$  do
14:    if  $i = force1.size()$  then
15:       $result[k] \leftarrow force2[j].p \cup force2[j].force + cont$ 
16:       $j \leftarrow j + 1$ 
17:    else if  $j = force2.size()$  then
18:       $result[k] \leftarrow force1[i].p \cup force1[i].force$ 
19:       $i \leftarrow i + 1$ 
20:    else if  $force1[i].p.y > force2[j].p.y$  then
21:       $result[k] \leftarrow force2[j].p \cup force2[j].force + cont$ 
22:       $j \leftarrow j + 1$ 
23:    else
24:       $result[k] \leftarrow force1[i].p \cup force1[i].force$ 
25:       $i \leftarrow i + 1$ 
26:       $cont \leftarrow cont + 1$ 
27:   $result \leftarrow preSort(result)$   $\triangleright$  Hacer presort en  $O(n)$ 
28:  return  $result$ 
```

4. Resultados

Se hicieron algunas pruebas comparando el tiempo de ejecución de los dos algoritmos anteriores, para ello se generaron conjuntos de puntos de distintos tamaños y se tomó el tiempo de ejecución de cada algoritmo con dichos conjuntos, a continuación se muestran los resultados.



```
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>main.exe 2 test_1.txt
Numero de puntos: 10

Tiempo transcurrido (algoritmo cuadratico): 0s.
Tiempo transcurrido (algoritmo divide y venceras): 0s.
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>
```

(a) Figura 1. Ejecución de ambos algoritmos con un conjunto de 10 puntos.

Para esta prueba, se usaron solamente 10 puntos, se puede apreciar que ambos algoritmos terminaron rápidamente ya que el conjunto de puntos era bastante pequeño.

```
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>main.exe 2 test_2.txt
Numero de puntos: 1000

Tiempo transcurrido (algoritmo cuadratico): 0.003s.
Tiempo transcurrido (algoritmo divide y venceras): 0.002s.
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>
```

(b) Figura 2. Ejecución de ambos algoritmos con un conjunto de 1000 puntos.

Para esta prueba, se usaron 1000 puntos, aquí se puede apreciar una pequeña diferencia de tiempos entre ambos algoritmos, pero en general siguen a la par.

```
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>main.exe 2 test_3.txt
Numero de puntos: 10000

Tiempo transcurrido (algoritmo cuadratico): 0.364s.
Tiempo transcurrido (algoritmo divide y venceras): 0.02s.
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>
```

(c) Figura 3. Ejecución de ambos algoritmos con un conjunto de 10000 puntos.

Para esta prueba, se usaron 10000 puntos, ya podemos notar una diferencia entre el tiempo de cómputo de ambos algoritmos, siendo que el de fuerza bruta estuvo cerca de medio segundo cuando el tiempo de cómputo del otro estaba en el orden de 10^{-2} .

```
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>main.exe 2 test_4.txt
Numero de puntos: 100000

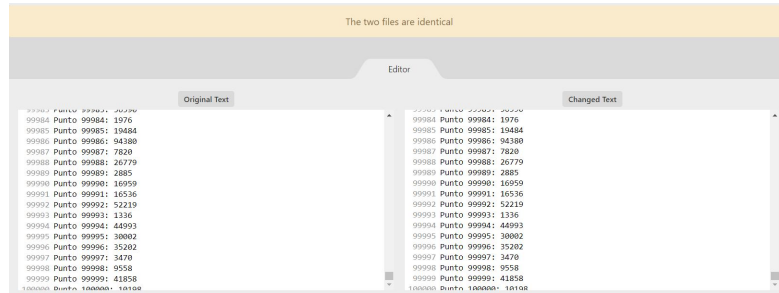
Tiempo transcurrido (algoritmo cuadratico): 36.093s.
Tiempo transcurrido (algoritmo divide y venceras): 0.196s.
D:\Github\Maestria\Semestre 3\Programacion y algoritmos 2\Tarea 6\Code>
```

(d) Figura 4. Ejecución de ambos algoritmos con un conjunto de 100000 puntos.

Finalmente en esta prueba se usaron 100000 puntos, aquí notamos una gran diferencia en el tiempo de procesamiento de ambos algoritmos, ya que el prime-

ro tardó alrededor de 40 seg. mientras que el segundo no tardó ni medio segundo.

Cabe destacar que también se tuvo que comparar los resultados arrojados por ambos algoritmos usando un software de comparación de textos, esto para ver que los resultados coincidían. Esto se muestra en la siguiente figura.



(e) Figura 5. Resultado de la comparación de los resultados arrojados por ambos algoritmos.