Tarea 3
Algoritmos

Emmanuel Peto Gutiérrez
José Luis Vázquez Lázaro

8 de octubre de 2022

Problema 1

a)

Se va a suponer que el algoritmo recibe un conjunto de puntos ordenado respecto a su coordenada x (en realidad se ordenan como un pre-procesamiento). A continuación se describen los pasos del algoritmo.

1) Si la lista de entrada tiene tamaño menor o igual a 1 entonces se devuelve la misma lista.

2) Partir el conjunto por la mitad en dos listas I (de izquierdo) y D (de derecho). En la lista I se encuentran los elementos que están a menos de la mitad de la lista de entrada mientras que en la lista D se encuentran los que estan a la mitad o más de la mitad.

3) Calcular recursivamente los puntos no dominados de I y D, y guardar el resultado en ndI y ndD, respectivamente.

4) Sea $pdmax$ el punto que tiene la coordenada y más grande en la lista ndD (los no dominados del lado derecho). Sea rv la lista que se va a devolver, inicialmente vacía.

5) Para cada punto pi de ndI: si la coordenada y de pi es mayor que la coordenada y de $pdmax$, entonces agregar pi a rv.

6) Agregar todos los puntos de ndD a rv.

7) Devolver rv.

En los algorimos NoDominados y NoDominadosAux se muestra un pseudocódigo que resuelve este problema. La función Append agrega un elemento al final de una lista y la función Extend concatena una lista con otra.

b)

Algoritmo 1: NoDominados(P)

Entrada: Un conjunto de puntos P donde no hay dos puntos con misma coordenada x ni misma coordenada y.

Salida: Puntos de P que no son dominados.

```
1 Ordenar P por su coordenada x;
2 NoDominadosAux(P);
```

Algoritmo 2: NoDominadosAux(P)

Entrada: Un conjunto de puntos P ordenados respecto a su coordenada x.

Salida: Puntos de P que no son dominados.

```
1  if |P| ≤ 1 then
2      return P;
3  I = ∅;
4  D = ∅;
5  mitad = ⌊|P|/2⌋;
6  for i = 0; i < |P|; i++ do
7      if i < mitad then
8          I.Append(P[i]);
9      else
10         D.Append(P[i]);
11 ndI = NoDominadosAux(I);
12 ndD = NoDominadosAux(D);
13 rv = ∅;
14 pdmax = ndD[0];
15 for pDer ∈ ndD do
16     if pdmax.y < pDer.y then
17         pdmax = pDer;
18 for pIzq ∈ ndI do
19     if pIzq.y > pdmax.y then
20         rv.Append(pIzq);
21 rv.Extend(ndD);
22 return rv;
```

Se demostrará por inducción sobre la cardinalidad de $|P|$ que el algoritmo NoDominadosAux(P) es correcto.

Caso base: $n \leq 1$.

Si el conjunto de puntos es vacío, se devuelve una lista vacía y por lo tanto el resultado es correcto. Si el conjunto de puntos tiene exactamente un elemento entonces no hay más puntos que lo dominen y se devuelve P. Estos casos base se tratan en las líneas 1 y 2 del código.

H.I. Supongamos que la función NoDominadosAux devuelve los puntos no dominados dado cualquier conjunto de puntos de cardinalidad menor a n.

P.D. La función NoDominadosAux devuelve el conjunto de puntos no dominados cuando $|P| = n$.

Entre las líneas 1 y 10 se divide al conjunto P en dos partes: I y D. La primera mitad se coloca en I y la segunda mitad en D. Como se está suponiendo que P está ordenado respecto a la entrada x y que no hay dos puntos que tengan la misma coordenada x, entonces en el plano todos los puntos de I quedan a la izquierda de todos los puntos de D.

En las líneas 11 y 12 se llama a la función NoDominadosAux con parámetro I y luego con parámetro D. Como I y D tienen cardinalidad menor a n, por hipótesis de inducción sabemos que la función NoDominadosAux devuelve exactamente los puntos no dominados de I y D, y los guarda en ndI y ndD, respectivamente. Ahora queda encontrar los puntos no dominados del conjunto completo (P) dado que tenemos los no dominados de su primera mitad y de su segunda mitad. Los puntos no dominados de P se guardarán en una lista rv que inicialmente está vacía.

Primero se analizará qué se hace con la lista ndI. Se sabe que todos los puntos de ndD tienen coordenada x mayor estricta que todos los puntos de ndI, así que posiblemente algunos de los puntos de ndD dominen a algunos puntos de ndI. Sea pI un punto de ndI; si pI es dominado por algún punto en ndD, entonces en particular es dominado por el punto en ndD que está más arriba (el que tiene la mayor coordenada y). Pero lo importante es que si pI no es dominado por el mayor (en su coordenada y) de ndD, entonces no es dominado por ningún punto de ndD. Con esto se tiene que basta con encontrar el punto con la coordenada y más grande en ndD para decidir cuáles puntos de ndI van a estar en la lista rv: se van a agregar solo aquellos puntos de ndI que estén por arriba de $pdmax$.

Sea $pdmax$ el punto con la coordenada y más alta en ndD (este punto se busca entre las líneas 14 y 17). Luego, para cada punto pI en ndI: si $pI.y > pdmax.y$ se agrega a la lista rv; si $pI.y \leq pdmax.y$ no se agrega. Esta parte del algoritmo se realiza entre las líneas 18 y 20.

Ahora observemos lo que pasa con el conjunto ndD. Como todos los puntos de ndD están a la derecha de todos los puntos de ndI, entonces no pueden ser dominados por ningún punto de ndI. Por hipótesis, se tiene que los puntos de ndD son "no dominados" en el lado derecho del plano, así que todos los puntos de ndD deben estar en la lista rv. Se agregan los puntos de ndD a rv en la línea 21 del pseudocódigo.

Con esto se tiene que rv tiene exactamente aquellos puntos de P que no son dominados por ningún otro punto de P. ■

c)

Ahora se va a analizar la complejidad.

En la función NoDominados se ordena el conjunto de puntos P respecto a x y luego se pasa este conjunto a NoDominadosAux, así que el algoritmo tiene complejidad $\Omega(n \log n)$. Para ver que *NoDominadosAux* corre en tiempo $O(n \log n)$ se verá que, salvo por las llamadas recursivas, se ejecutan $O(n)$ operaciones.

- Partir P en las sublistas I y D toma tiempo $O(n)$, pues solo se hace una pasada a la lista P.
- Encontrar al máximo en y ($pdmax$) de la lista ndD toma tiempo $O(n)$.
- Agregar los elementos de ndI a rv que tengan coordenada y más grande que $pdmax$ toma tiempo $O(n)$.
- Agregar todos los elementos de ndD a rv toma tiempo $O(n)$.

Con esto se tiene que en el algoritmo se realizan un número lineal de operaciones más dos llamadas recursivas a NoDominadosAux con ejemplares de la mitad del tamaño. Así que el tiempo de ejecución de NoDominadosAux satisface la recurrencia $T(n) = 2T(n/2) + O(n)$ y la solución de esta recurrencia es $O(n \log n)$. ■