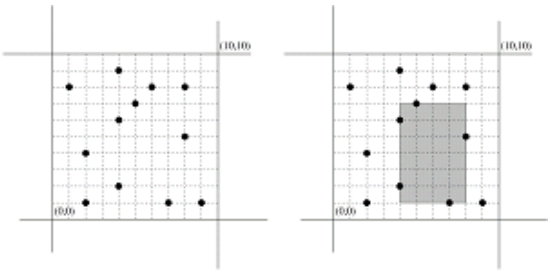
La casa más grande

Manuel Abellanas Oar, Ángel Herranz Nieva
Facultad de Informática, Universidad Politécnica de Madrid

<{mabellanas, aherranz}@fi.upm.es>

El enunciado de este problema apareció en el número 176 de ***Novática*** (julio-agosto 2005, p. 74). Es el problema B de los planteados en el III Concurso Universitario de la Comunidad Autónoma de Madrid (CUPCAM 2005), del que ATI fue entidad colaboradora.

Recordemos que el problema consiste en encontrar el tamaño del mayor rectángulo vacío de árboles cuyos lados son paralelos a los lados de la parcela, que es cuadrada. Es importante también recordar que los árboles en la parcela se sitúan en puntos de coordenadas enteras.



Parcela de 10 x 10 Rectángulo más grande

Se describen dos soluciones diferentes. La primera es ligeramente más sencilla de implementar por lo que un equipo en competición podría planteársela como primera opción. La segunda, aunque en el peor caso tiene una complejidad similar a la primera, tiene la ventaja de que su complejidad depende del número de árboles y no del tamaño de la parcela. Así, en los casos en los que hay pocos árboles, comparado su número con el tamaño de la parcela, la segunda solución es mucho más eficiente.

1. ¡Rápido!

En un concurso de programación es bastante habitual que los equipos ataquen algunos problemas con soluciones que suponen un tiempo de desarrollo lo más corto posible. En este caso, una solución inmediata consiste en analizar todos los rectángulos contenidos en la parcela cuyos vértices tienen coordenadas enteras. Cada rectángulo está determinado por dos de sus vértices diagonalmente opuestos. Así, para cada par de puntos de coordenadas enteras contenidos en la parcela, se evalúa si el rectángulo que determinan está vacío de árboles. En caso de contener algún árbol, se descarta pasando a analizar el siguiente par. Si está vacío, se calcula su área y se compara con el área máxima obtenida hasta el momento. En caso de ser mayor, se sustituye, pasando a ser ésta la nueva solución provisional.

Como se ve, el algoritmo requiere de un triple bucle anidado: dos para recorrer todos los pares de puntos de coordenadas enteras dentro de la parcela y el tercero para verificar si el rectángulo es válido. Nótese que cada punto está determinado por dos coordenadas enteras que varían entre 0 y n (siendo n el tamaño del lado de la parcela), por lo que, en realidad son 5 bucles anidados. Cuatro de ellos varían entre 0 y n, y el quinto, más interno, entre 1 y el número t de árboles.

Nosotros optamos por presentar una variante, de dificultad análoga, que es la de analizar todos los rectángulos posibles que se pueden obtener con la distribución de árboles. Para ello se generan todos los rectángulos posibles y se filtran con el criterio de no contener árboles para, posteriormente, quedarse con aquél que tenga el área mayor. La solución en Haskell 98 es bastante concisa:

```
import List

-- Representación de puntos y rectángulos
type Punto = (Int,Int)
type Rectangulo = (Punto,Punto)

-- Comprueba la pertenencia de un punto a
-- un rectángulo
enRectangulo :: Rectangulo -> Punto -> Bool
enRectangulo ((x,y),(x'',y'')) (x',y') =
  x < x' && x' < x'' && y < y' && y' < y''

-- Comprueba la no pertenecia de ningún
-- punto a un rectángulo
sinPuntos :: [Punto] -> Rectangulo -> Bool
sinPuntos puntos rect =
  not (any (enRectangulo rect) puntos)

-- Genera una rejilla a partir de puntos
rejilla :: Int -> [Punto] -> [(Int,Int)]
rejilla n puntos =
  [(x,y) | x <- hs, y <- vs]
  where
    hs =
      nub (0 : n : [x | (x,_) <- puntos])
    vs =
      nub (0 : n : [y | (_,y) <- puntos])

-- Genera rectángulos libres de puntos
rectangulos ::
  Int -> [Punto] -> [Rectangulo]
rectangulos n puntos =
  filter (sinPuntos puntos) todos
  where todos = [(p,q) | p@(x,y) <- grid,
                         q@(x',y') <- grid,
                         x < x', y < y']
        grid = rejilla n puntos

-- Area de un rectángulo
area :: Rectangulo -> Int
area ((x,y),(x',y')) = (x' - x) * (y' - y)

-- El area más grande de un conjunto de
-- rectángulos
masGrande :: [Rectangulo] -> Int
masGrande =
  foldl (\a r -> max a (area r)) 0

-- Lee puntos de la entrada estándar
leePuntos :: Int -> IO [Punto]
leePuntos n
  | n > 0 =
      do line <- getLine
         let [s1,s2] = words line
         let x = read s1 :: Int
         let y = read s2 :: Int
         puntos <- leePuntos (n-1)
         return ((x,y) : puntos)
  | otherwise =
      return []

-- Función de ordenación para rectángulos
rectOrder ::
  Rectangulo -> Rectangulo -> Ordering
rectOrder r s = compare (area s) (area r)
```

```
main :: IO ()
main =
```

```
do line1 <- getLine
   let n = read line1 :: Int
   if (n > 0)
     then
       do line2 <- getLine
          let t = read line2 :: Int
          puntos <- leePuntos t
          let rects = rectangulos n puntos
          –print (sortBy rectOrder rects)
          print (masGrande rects)
          main
     else putStr ""
```

2. ¡Eficaz!

Si se presta más atención al problema no resulta complicado descubrir algunas propiedades de la solución esperada que pueden ayudarnos a escribir un programa más eficaz en general y más eficiente cuando el número de árboles no es elevado.

El rectángulo solución queda delimitado por cuatro árboles, uno en cada uno de sus lados (ó k árboles y 4-k lados de la parcela, $0 \leq k \leq 4$). Así, tomando de cuatro en cuatro los árboles (o lados de la parcela), podemos obtener un conjunto de rectángulos candidatos entre los que se encuentra la solución:

Para cada cuatro árboles (o lados de la parcela) seleccionados basta con hacer lo siguiente:

- Analizar si están en posición adecuada para determinar un rectángulo: el más alto y el más bajo no deben ser ni el más a la izquierda ni el más a la derecha. En caso negativo pasar a la siguiente selección.
- Analizar si el rectángulo que determinan está vacío de árboles. En caso negativo pasar a la siguiente selección.
- Calcular el área del rectángulo y compararla con la solución provisional acumulada hasta el momento, sustituyendo el valor en caso de ser mayor el nuevo.

Obsérvese que el algoritmo consta de cinco bucles anidados, todos ellos variando entre 1 y el número t de árboles.

3. Más eficiencia

Existen soluciones más eficientes cuya descripción excede el ámbito de esta sección. Al lector interesado se le sugiere consultar dentro del área de la geometría computacional los temas *puntos maximales orientados* y *geometría de los rectángulos isotéticos*. El problema está también relacionado con las búsquedas en rangos del área de bases de datos.