Universidad Rey Juan Carlos

Grado en Matemáticas

Práctica 9

Geometría Computacional

Autores: Guillermo Grande Santi y

Alejandro López Adrados

Abril, 2022

Índice

1 Objetivos

El objetivo de esta práctica es elegir un mapa cualquiera y programar una coloración del mismo. Para ello hemos implementado el algoritmo de coloración de Welsh y Powell.Re specto a la selección del mapa para aplicar dicho algoritmo, hemos importado la mariposa de la práctica 1, la cual se trataba de un mapa ya triangulado, lo que nos facilita el trabajo, cabe recordar que esta mariposa se componía de 63 puntos los cuales se introducen en funciones de x e y para obtener como resultado tal dibujo.

2 Metodología

La primera parte del código es la implementación de la **mariposa** según la práctica 1, describiendo sus puntos, añadiendo sus vectores de triangulación manualmente y pintándola en color blanco por pantalla.
La matriz vecinos se encargará de recoger todos estos vértices de la triangulación, mientras que la matriz **vértices** se formará con tres columnas distinguidas. En la primera columna se guarda el número del vector, en la segunda columna se guarda el grado de ese vértice (apariciones en la matriz vecinos) y en la tercera columna se guarda el color del vértice una vez sea coloreado.
Una vez tenemos esto, se ordena la segunda columna de vértices (grado del vértice) con **quicksort** de mayor a menor, que se llamará aparicionesOrdenadas. Entonces, con la ayuda de esta matriz se ordena la matriz vértices al completo, tal y como se hizo en la práctica 7 al tratarse de una matriz bidimensional.
En este punto, se crean varias matrices. VerticesGlobal guardará todos los vértices que tenemos (ya que en la siguiente función los vértices se irán modificando hasta tener una matriz vacía), **puntosColoreados** guardará los vértices (su número) que han sido coloreados por un color en la ejecución de la función y **conexiones** guardará todos los vértices (su número) que sean conexiones de dichos puntos guardados en puntosColoreados. Así, se tendrá control sobre

cualquier punto que esté conectado por un arista a cualquier punto coloreado. Entonces comienza la función **coloracion**, que comienza creando una función interna llamada **estaEnConexiones()** que, en caso de que nos devuelva 1 nos indicará que el vector que le hemos pasado como parámetro está ya incluido en la matriz conexiones. En caso contrario (devuelve 0) nos indicará que el punto no está incluido en conexiones. Como primeros pasos, se guardan las dimensiones de la matriz vértices para poder usarlo en los límites de los bucles. Se empezará entonces coloreado el primer vértice de la lista de vértices, y posteriormente buscaremos todas las conexiones que tiene dicho punto con el resto de vértices. Para ello se comprabará cada fila del vector vecinos en busca del vértice prefijado anteriormente y se guardarán en la matriz conexiones los vértices que le acompañen (sólo si estos no han sido incluido anteriormente en la matriz conexiones).
Una vez tenemos todos los vértices con conexión se buscarán en la matriz vértices cuál es la primera aparición **sin conexión**. Una vez se ha encontrado, se coloreará del mismo color que el anterior y se añadirá a la matriz puntosColoreados. Tras esto se repite el proceso buscando las conexiones de este nuevo vértices coloreado hasta llegar al final de la matriz vértices.

En este punto tendremos varios vértices coloreados del mismo color sin conexiones entre ellos. Así, estos vértices ya coloreados se **eliminarán** de la matriz, y se realizarán llamadas recursivas con esta nueva lista de vértices y el **siguiente color** hasta llegar a tener una matriz vacía. Entonces todos los puntos habrán sido coloreados de manera correcta, finalizando así el algoritmo.

3 Conclusiones

Tras esta práctica hemos aprendido otro tipo de algoritmo, lo cual viene siendo costumbre en estas últimas prácticas. Sin embargo, cada una tiene algo diferente, y en este caso ha sido el uso de la recursividad unido a un factor ya repetido en anteriores prácticas como es la utilización de prácticas anteriores.

En este caso concreto, hemos utilizado un fragmento de la práctica 7 que a su vez utilizaba un fragmento de algoritmos de ordenación rápidos, además de la utilización de la mariposa de la práctica 1, lo cual nos ha servido como mapa. En esta práctica cabe resaltar lo costoso del entendimiento de dicho algoritmo lo que nos ha llevado a emplear un poco más tiempo en pensar su realización que por ejemplo la anterior que el algoritmo era más claro.

4 Anexos

4.1 Código del ejercicio

```
#Algoritmo de ordenamiento
quicksort <- function(x)
{
  #Caso base
  if (length(x) <= 1) {
    return(x)
  }
  #Inicializamos un array que guardará todos los valores menos la mediana
  resto <- c()
  #Elegimos la mediana del vector
  mediana <- length(x) %/% 2
  #Se elige el pivote
  pivote <- x[mediana]
  #Bucle que inserta en resto hasta la posición mediana-1
  for (i in 1:mediana - 1) {
    resto[i] <- x[i]
  }
  longitud <- length(x)
  siguiente_mediana <- mediana + 1
```

```
  #Bucle que inserta en resto desde la posición mediana+1
  for (i in siguiente_mediana:longitud) {
    resto[i - 1] <- x[i]
  }
  #Recursividad sobre el vector resto:
  #Parte izquierda del árbol (menores)
  L1 <- quicksort(resto[resto > pivote])
  #Parte derecha del árbol (mayores)
  L3 <- quicksort(resto[resto <= pivote])
  #Se devuelve la concatenación de parte izquierda-pivote-parte derecha
  return(c(L1, pivote, L3))
}



#Creación de todos los puntos de la mariposa
t<-seq(0,2*pi,0.1)
x<-sin(t)*(exp(1)^cos(t)-2*cos(4*t)-sin(t/12)^5)
y<-cos(t)*(exp(1)^cos(t)-2*cos(4*t)-sin(t/12)^5)
plot(-4:4,-4:4)
areaTotal<-0
#polygon(x,y, col = "white", border = "white")

#Triangulación de la mariposa de Práctica1

#ALA DERECHA
#v1<-c(1, 34, 35)
v2<-c(3, 35, 14)
v3<-c(29, 35, 14)
v4<-c(7, 6, 5)
v5<-c(8, 7, 5)
```

```
v6<-c(9, 8, 5)
v7<-c(10, 9, 5)
v8<-c(11, 10, 5)
v9<-c(12, 11, 5)
v10<-c(13, 12, 5)
v11<-c(14, 13, 5)
v12<-c(4, 14, 5)
v13<-c(3, 14, 4)
#ALA IZQUIERDA
v14<-c(52, 60, 51)
v15<-c(60, 53, 52)
v16<-c(60, 54, 53)
v17<-c(60, 55, 56)
v18<-c(60, 55, 54)
v19<-c(60, 55, 56)
v20<-c(60, 57, 56)
v21<-c(60, 57, 58)
v22<-c(60, 59, 58)
v23<-c(61, 60, 51)
v24<-c(61, 62, 51)
v25<-c(30, 62, 51)
v26<-c(30, 29, 51)
v27<-c(30, 62, 31)
#CABEZA
v28<-c(31, 33, 32)
v29<-c(1, 2, 3)
v30<-c(3, 1, 33)
v31<-c(31, 33, 1)
v32<-c(31, 63, 62)
v33<-c(31, 63, 1)
```

```
#MINI ALA IZQUIERDA
v34<-c(15, 16, 17)
v35<-c(17, 18, 19)
v36<-c(15, 19, 17)
v37<-c(15, 19, 29)
#MINI ALA DERECHA
v38<-c(50, 49, 48)
v39<-c(48, 47, 46)
v40<-c(46, 50, 48)
v41<-c(50, 29, 46)
#ALA INFERIOR IZQUIERDA
v42<-c(40, 42, 44)
v43<-c(40, 42, 44)
v44<-c(40, 38, 44)
v45<-c(38, 44, 36)
v46<-c(42, 43, 44)
v47<-c(38, 39, 40)
v48<-c(38, 37, 36)
v49<-c(40, 41, 42)
v50<-c(36, 45, 44)
v51<-c(36, 45, 29)
#ALA INFERIOR DERECHA
v52<-c(25, 24, 23)
v53<-c(21, 23, 25)
v54<-c(21, 23, 22)
v55<-c(25, 26, 27)
v56<-c(21, 25, 27)
v57<-c(21, 27, 28)
v58<-c(20, 21, 28)
v59<-c(28, 29, 20)
```

```
#Introducimos todos los vectores en una matriz
vecinos<-rbind(v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15,v16
                ,v17,v18,v19,v20,v21,v22,v23,v24,v25,v26,v27,v28,
                v29,v30,v31,v32,v33,v34,v35,v36,v37,v38,v39,v40,
                v41,v42,v43,v44,v45,v46,v47,v48,v49,v50,v51,v52,
                v53,v54,v55,v56,v57,v58,v59)

#Se pinta la mariposa
for(i in 1:58){
  polygon(c(x[vecinos[i,1]],x[vecinos[i,2]],x[vecinos[i,3]]),c(y[vecinos[i,1]],y[vecinos[i,2
}

#Estos vertices serán los puntos del mapa a colorear
#Cambiarlos para hacer el programa con otro polígono cualquiera.
vertices<-matrix(nrow=63,ncol=3)
i<-1
#Vertices será una matriz de 3 columnas.
#En la primera columna se guarda el número del vector
#En la segunda columna se guarda el grado de ese vértice (apariciones en vecinos)
#En la tercera columna se acabará guardando el color del vértice una vez sea coloreado.
for(i in 1:63){
  vertices[i,1]<-i
  vertices[i,2]<-0
}
i<-1
j<-1
h<-1
for(h in 1:63){
  for(i in 1:58){
    for(j in 1:3){
```

```
      if (vecinos[i,j]==h){
        #Aumenta en 1 las apariciones
        vertices[h,2]<-vertices[h,2]+1
      }
     }
  }
}
i<-1
dimension<-dim(vertices)
n<-dimension[1]
while(i!=n){
    if (vertices[i,2]<=1){
      vertices <- vertices[-i, ]
    }
  dimension<-dim(vertices)
  n<-dimension[1]
  i<-i+1
}

#Se ordenan la segunda columna de vertices (grado del vértice)
#con quicksort de mayor a menor.
aparicionesOrdenadas <- quicksort(vertices[,2])

#Con las apariciones ordenadas, se ordena la
#matriz vértices al completo.
h <- 1
k <- 1
dimension2 <- dim(t(aparicionesOrdenadas))
for (h in 1:dimension2[2]) {
  while (aparicionesOrdenadas[h] != vertices[k,2]) {
```

```
    k = k + 1
  }
  aux <- vertices[h, ]
  vertices[h, 1] <- vertices[k, 1]
  vertices[h, 2] <- vertices[k, 2]
  #vertices[h, 3] <- vertices[k, 3]
  vertices[k, ] <- aux
  if(h!=dimension2[2]){
  if(aparicionesOrdenadas[h]!=aparicionesOrdenadas[h+1]){
    k <- 2
  }
}
}

#Guardamos la matriz de vertices, ya que se va a modificar en la funcion
verticesGlobal<-vertices

#Creamos una matriz de conexiones de los puntos coloreados
conexiones <- matrix(nrow = 63, ncol = 1)
puntosColoreados <- matrix(nrow = 63, ncol = 1)
dimensionV<-dim(vertices)


coloracion <- function(vertices, color) {
    #funcion para ver si un elemento esta en la matriz conexiones
  estaEnConexiones<-function(elemento){
    c<-0
    for(a in conexiones){
      a
      if(!is.na(a)){
```

```
      if(a==elemento){
        c<-1
      }
    }
  }
  c
}
dimensionV<-dim(vertices)
n<-dimensionV[1]
puntosColoreados <- matrix(nrow = dimensionV[1], ncol = 1)
#Primer vertice coloreado
vertices[1, 3] <- color
conexiones <- matrix(nrow = dimensionV[1], ncol = 1)
indice <- 1
indice2 <- 2
puntosColoreados[1] <- vertices[1, 1]
indice <- indice + 1
s <- 1
j <- 1
#Se recorren los vertices buscando conexiones con los puntos coloreados
#Si hay conexion y ese vertice no esta guardado todavia se guarda
while (!is.na(puntosColoreados[s])) {
  for (i in 1:58) {
    if (vecinos[i, 1] == puntosColoreados[s]) {
      if (estaEnConexiones(vecinos[i, 2]) == 0) {
        conexiones[j] <- vecinos[i, 2]
        j <- j + 1
      }
      if (estaEnConexiones(vecinos[i, 3]) == 0) {
        conexiones[j] <- vecinos[i, 3]
```

```
        j <- j + 1
      }
    } else if (vecinos[i, 2] == puntosColoreados[s]) {
      if (estaEnConexiones(vecinos[i, 1]) == 0) {
        conexiones[j] <- vecinos[i, 1]
        j <- j + 1
      }
      if (estaEnConexiones(vecinos[i, 3]) == 0) {
        conexiones[j] <- vecinos[i, 3]
        j <- j + 1
      }
    } else if (vecinos[i, 3] == puntosColoreados[s]) {
      if (estaEnConexiones(vecinos[i, 1]) == 0) {
        conexiones[j] <- vecinos[i, 1]
        j <- j + 1
      }
      if (estaEnConexiones(vecinos[i, 2]) == 0) {
        conexiones[j] <- vecinos[i, 2]
        j <- j + 1
      }
    }
  }
  #Una vez guardadas todas las conexiones se buscan la primera
  #aparicion de un vertice que no tenga conexiones
  while (estaEnConexiones(vertices[indice, 1]) == 1) {
    indice <- indice + 1
    #Hasta que se recorre la matriz entera
    if (indice>=n)
      break
  }
```

```
  if (indice<n) {
    puntosColoreados[indice2] <- vertices[indice, 1]
    vertices[indice, 3] <- color
    indice <- indice + 1
    indice2 <- indice2 + 1
  }
  #Se vuelve a buscar con dicho vertice
  s <- s + 1
}
for(f in 1:dimensionV[1]){
  if(!is.na(vertices[f,3])){
    points(x[vertices[f,1]],y[vertices[f,1]],col=color,pch=19)
  }
}
p<-1
dimensionJ<-dim(vertices)
#Se eliminan los vertices ya coloreado de la matriz vertices
if(dimensionJ[1]>0){
  while(p!=dimensionJ[1]){
      if(!is.na(vertices[p,3])){
          vertices <- vertices[-p,]
      p<-p-1
      }
      dimensionJ<-dim(vertices)
      if(is.null(dimensionJ[1])){break}
      p<-p+1
  }
  color <- color + 1
  #Llamadas recursivas a la funcion hasta que la matriz vertices es vacia
  if(!is.null(dimensionJ[1])){
```

```
        coloracion(vertices,color)
    }
  }
}

coloracion(vertices,2)
```