Divide y vencerás

17 de enero de 2023

Analisis y Diseño de Algoritmos

Herrera Guadarrama Juan Pablo

0.Divide y vencerás

- Recursiva
- Descompone el problema original en subproblemas que son mas fáciles de resolver
- Instancias mas sencillas
- Árbol de invocaciones
- En el peor de los casos con una profundidad alta

Análisis y diseño

Una breve explicación…
El programa a grandes rasgos gira una imagen, con la ayuda de recursividad y la técnica de divide y vencerás.
Para una imagen .bmp cuadrada, donde el lado=potencia de 2 es decir (2*2,4*4,16*16, 2^n* 2^n)
Lo que se hace es ir reduciendo la imagen si se tiene una imagen de 16*16, segmenta la imagen hasta tener una de 2*2 y ahí es cuando trabaja las operaciones de mover los pixeles para que la imagen quede rotada esto con la ayuda de recursión la parte de divide y vencerás es la segmentación de la imagen si se tenia una matriz de 16*16 se redujo a pequeñas imágenes de 2*2

Su complejidad es *NlogN*

Código fuente

Usando programación estructurada

Main.h

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <stdint.h>
4. #include <string.h>
5. #include <math.h>
6.
7.
8. //cambiar numero para seleccionar distintas  resoluciones
9. int n = 1024;
10.    char *entrada="ejem2_c.bmp";
11.    char *salida="ImagenR2.bmp";
12.
13.    /* run this program using the console pauser or add your own getch,
   system("pause") or input loop */
14.
15.    int main(int argc, char *argv[]) {
16.
17.          FILE* archvLec;
18.          FILE* archvEsc;
19.          int tam, tam2;//ancho y alto de la imagen
20.          int direccion;//direccion donde se guardan los datos de las matrices
21.          int leidos = 0;
22.          int i=0, j=0;;
```

```
23.         unsigned char *ap;
24.         unsigned char buffer[30];
25.         unsigned char temporal;
26.
27.         //abre el archivo
28.       archvLec = fopen(entrada, "rb");
29.       if( archvLec == NULL )
30.        {
31.              perror("Error: no se abrio el archivo\n");
32.              exit (0);
33.        }
34.
35.
36.       //lee el header del archivo bmp
37.       for(;i<30;i++)
38.       {
39.         buffer[i]= fgetc(archvLec);//lee byte por byte
40.         //guarda la direccion de la matriz
41.         if( i>=10 && i<=13 )
42.         {
43.               ap=(unsigned char *)&direccion;
44.               ap[i-10]=buffer[i];
45.               }
46.
47.         //guarda el tam (ancho) de la matriz
48.         if( i>=18 && i<=21 )
49.         {
50.               ap=(unsigned char *)&tam;
51.               ap[i-18]=buffer[i];
52.               }
53.               //guarda el tam (alto) de la matriz
54.         if( i>=22 && i<=25 )
55.         {
56.               ap=(unsigned char *)&tam2;
57.               ap[i-22]=buffer[i];
58.               }
59.         }
60.         printf("La anchura es:%d Y la altura es:%d\n", tam, tam2);
61.
62.       //comprobaciones de tamanio de la imagen
63.       if( tam != tam2)//comprueba que es cuadrada
64.       {
65.         printf("Error ingrese una imagen de resolucion cuadrada");
66.         exit(0);
67.         }
68.         for(i=0; i<34;i++)//comprueba que sea una potencia de 2, menor a 2^32
69.         {
70.               if( (double)tam == pow((double)2, (double)i) )
71.               {
72.                     break;
73.               }
74.               if( i == 33 )
75.               {
76.                     printf("Tu archivo no es potencia de 2");
77.                     exit(0);
78.               }
79.         }
80.         printf("Archivo aceptado. Prosiguiendo\n");
81.
82.
83.         //creacion de la matriz
84.         unsigned char **matrizB;//B
85.         unsigned char **matrizG;//G
86.         unsigned char **matrizR;//R
87.         matrizB = asignaMemMatrizCuadrada(matrizB, tam);
88.         matrizG = asignaMemMatrizCuadrada(matrizG, tam);
```

```
89.         matrizR = asignaMemMatrizCuadrada(matrizR, tam);
90.
91.
92.         rewind(archvLec);//regresa al inicio del archivo
93.
94.       //for que mueve al comienzo de los datos de las matrices para despues
   leer de las matrices
95.       for(i=0; i<direccion; i++)
96.         {
97.         temporal=fgetc(archvLec);
98.         }
99.
100.        //empieza a guardar los datos en las matrices en memoria
101.        for(i=0; i<tam; i++)//filas
102.        {
103.              for(j=0; j<tam; j++)//columnas
104.              {
105.                    matrizB[i][j]=fgetc(archvLec);
106.                    matrizG[i][j]=fgetc(archvLec);
107.                    matrizR[i][j]=fgetc(archvLec);
108.                    temporal=fgetc(archvLec);//lee basura (0xff)
109.              }
110.        }
111.
112.      //gira las matrices usando divide y venceras
113.      matrizB = girar(matrizB,tam);
114.      matrizG = girar(matrizG,tam);
115.      matrizR = girar(matrizR,tam);
116.
117.      /*generacion el nuevo archivo*/
118.      rewind(archvLec);//regresa al inicio del archivo
119.      archvEsc = fopen(salida, "wb");
120.
121.        //lee la cabecera vieja y escribe la nueva
122.        int escritos;
123.      for(i=0; i<direccion; i++)
124.        {
125.        temporal=fgetc(archvLec);//lee del archivo de entrada byte a byte
126.        escritos = fwrite(&temporal, sizeof(unsigned char), 1,
   archvEsc);//escribe en el archivo byte por byte conforme lee
127.        }
128.        temporal=0xff;//variable 0xff que separa los datos de las matrices del
   archivo bmp
129.        for(i=0; i<tam; i++)//filas
130.        {
131.              for(j=0; j<tam; j++)//columnas
132.              {
133.                    escritos=fwrite(&matrizB[i][j], sizeof(char), 1,
   archvEsc);//escribe el dato de la matriz Blue
134.                    escritos=fwrite(&matrizG[i][j], sizeof(char), 1,
   archvEsc);//escribe el dato de la matriz Green
135.                    escritos=fwrite(&matrizR[i][j], sizeof(char), 1,
   archvEsc);//escribe el dato de la matriz Red
136.                    escritos=fwrite(&temporal, sizeof(char), 1,
   archvEsc);//escribe la constante 0xff
137.              }
138.        }
139.
140.        //libera la memoria de la matrices usadas
141.        for(i=0; i<tam; i++)
142.        {
143.              free(matrizB[i]);
144.              free(matrizG[i]);
145.              free(matrizR[i]);
146.        }
147.        free(matrizB);
```

```
148.        free(matrizG);
149.        free(matrizR);
150.
151.        //termina y cierra los archivos
152.
153.        //fprintf( "%d,\n", n);
154.
155.        fclose(archvEsc);
156.        fclose(archvLec);
157.        return 0;
158.    }
```

funciones_P1.h

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <math.h>


//asigna memoria dinamica a una matriz cuadrada tamXtam
unsigned char** asignaMemMatrizCuadrada(unsigned char** matriz, int tam)
{
    int i = 0;
    matriz = (unsigned char **)malloc(tam*sizeof(unsigned char*));

    if( matriz == NULL )
    {
        perror("Error al asignar memoria de filas");
        exit(0);
    }
    for(i=0 ; i<tam ;i++)
    {
        matriz[i]=(unsigned char*)malloc(tam*sizeof(unsigned char));
        if( matriz[i] == NULL)
        {
            perror("Error al asignar memoria de columnas");
        }
    }
    return matriz;
}

//muestra la matriz en pantalla
void muestraMatriz(unsigned char**matriz, int tam)
{
    int i,j;
    for(i=0; i<tam;i++)//recorre filas
    {
        for(j=0; j<tam;j++)//recorre columnas
        {
            printf("[%.2x] ", matriz[i][j]);
        }
        puts("");
    }
}

//funcion que gira la matriz
unsigned char** girar(unsigned char **matriz,int ancho){

      if( ancho == 2 ){
      //caso base
        unsigned char aux;
            aux = matriz[0][0];//guarda valor en variable temporal
```

```
        //gira la matriz de 2x2

            matriz[0][0] = matriz[0][1];
            matriz[0][1] = matriz[1][1];
            matriz[1][1] = matriz[1][0];
            matriz[1][0] = aux;

            return matriz;
      }
      else
      {
        //declaracion de variables
        unsigned char **miniMatriz;
        unsigned char **matriz1;
        unsigned char **matriz2;
        unsigned char **matriz3;
        unsigned char **matriz4;

        //asigna memoria a las matrices
            miniMatriz = asignaMemMatrizCuadrada(miniMatriz, ancho/2);
            matriz1 =asignaMemMatrizCuadrada(matriz1,ancho/2);
            matriz2 =asignaMemMatrizCuadrada(matriz2,ancho/2);
            matriz3 =asignaMemMatrizCuadrada(matriz3,ancho/2);
            matriz4 =asignaMemMatrizCuadrada(matriz4,ancho/2);

        //copia los datos de las dos matrices internas superiores
            copiaDosMatr(matriz, matriz1, matriz2, ancho/2);
        //copia los datos de las dos matrices internas inferiores
            copiaDosMatr(matriz+ancho/2, matriz3, matriz4,ancho/2);

        //llamada recursiva a la funcion girar enviando las cuatro matrices que
conforman a matriz
            matriz1=girar(matriz1, ancho/2);
            matriz2=girar(matriz2, ancho/2);
            matriz3=girar(matriz3, ancho/2);
            matriz4=girar(matriz4, ancho/2);

        //copia en una matriz aux los datos de matriz1
      copia(miniMatriz, matriz1, ancho/2);

        //gira las matrices con los datos internos ya girados
            copia(matriz1, matriz2, ancho/2);
            copia(matriz2, matriz4, ancho/2);
            copia(matriz4, matriz3, ancho/2);
            copia(matriz3, miniMatriz, ancho/2);


            //reasigna a la matriz original

        //dadas matriz1 y matriz2 reasigna los datos en la parte superior de matriz
            copiaDosMatrInv(matriz, matriz1, matriz2, ancho/2);

        //dadas matriz3 y matriz4 reasigna los datos en la parte inferior de matriz
            copiaDosMatrInv(matriz+ancho/2, matriz3, matriz4, ancho/2);

        //libera la memoria de matriz
            free (miniMatriz);
            free (matriz1);
            free (matriz2);
            free (matriz3);
            free (matriz4);
      }
      return matriz;
}
```

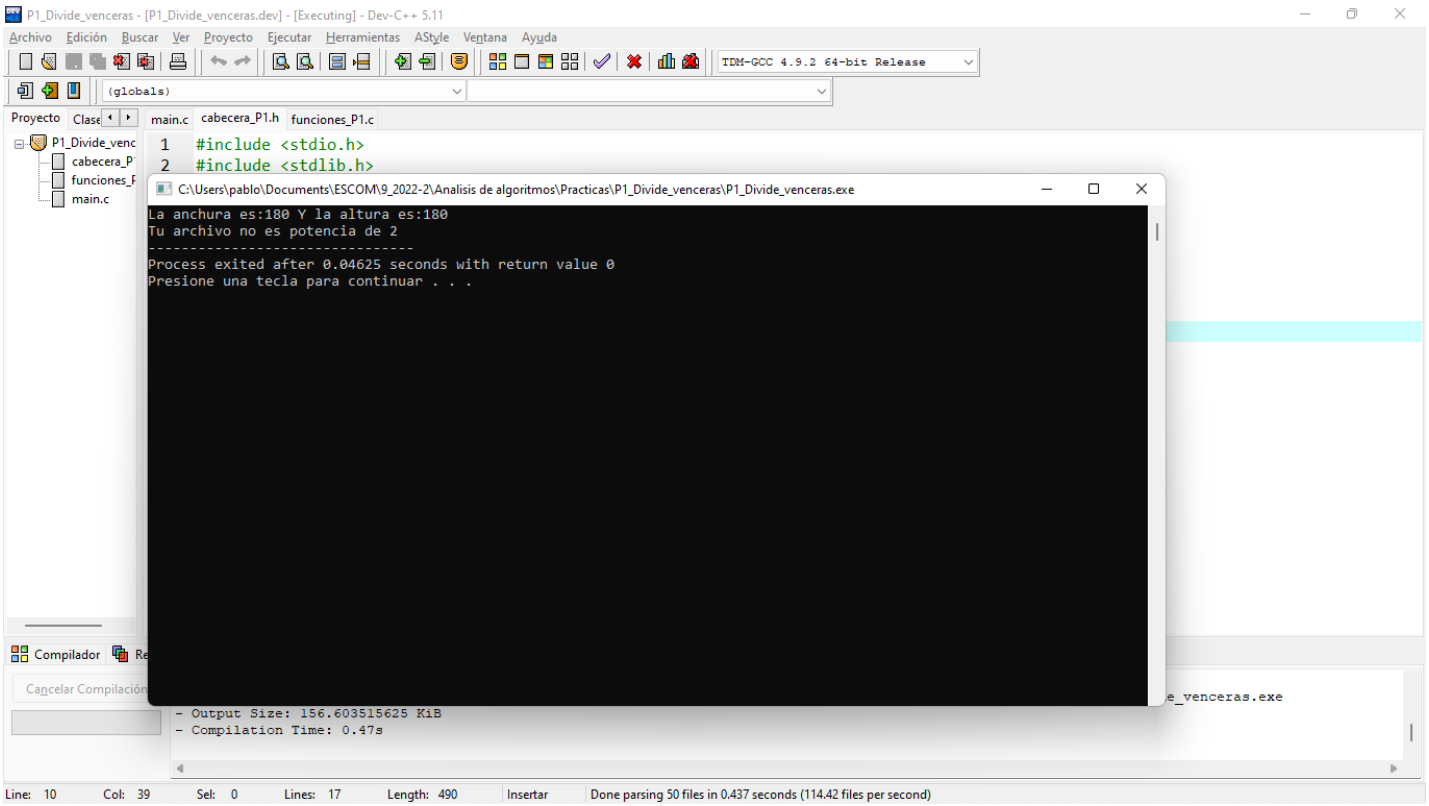
```
void copia(unsigned char **matrizX,unsigned char**matrizY, int ancho)
{
	int i,j;


	for(i=0; i<ancho; i++)//recorre filas
	{
		for(j=0; j<ancho; j++)//recorre columasn
		{

			matrizX[i][j]=matrizY[i][j]; //copia el contenido

		}

	}
}

//copiados matrices debido a que en matrizO se encuentran los apunatadores a las filas
de tamanio 0 a ancho
//( que tiene los datos de la matriz de izquierda (matriz 1) y la matriz
derecha(matri2))
void copiaDosMatr(unsigned char **matrizO,unsigned char**matrizIzq, unsigned
char**matrizDer, int ancho)
{
	int i,j;

	for(i=0; i<ancho; i++)//filas
	{

		for(j=0; j<ancho; j++)//columnas
		{

			matrizIzq[i][j] = matrizO[i][j];
			matrizDer[i][j] = matrizO[i][j+ancho];

		}

	}
}

//dadas dos matrices copia en matrizO los datos de ambas, debido a que en matrizD se
encuentran los apunatadores
//a las filas de tamanio 0 a ancho (que tiene los datos de la matriz de izquierda
(matriz 1) y la matriz derecha(matri2))
void copiaDosMatrInv(unsigned char **matrizD,unsigned char**matrizIzq, unsigned
char**matrizDer, int ancho)
{
	int i,j;

	for(i=0; i<ancho; i++)
	{

		for(j=0; j<ancho; j++)
		{

			matrizD[i][j]=matrizIzq[i][j];
			matrizD[i][j+ancho]=matrizDer[i][j];

		}

	}
	}
```

Cabecera_P1.h

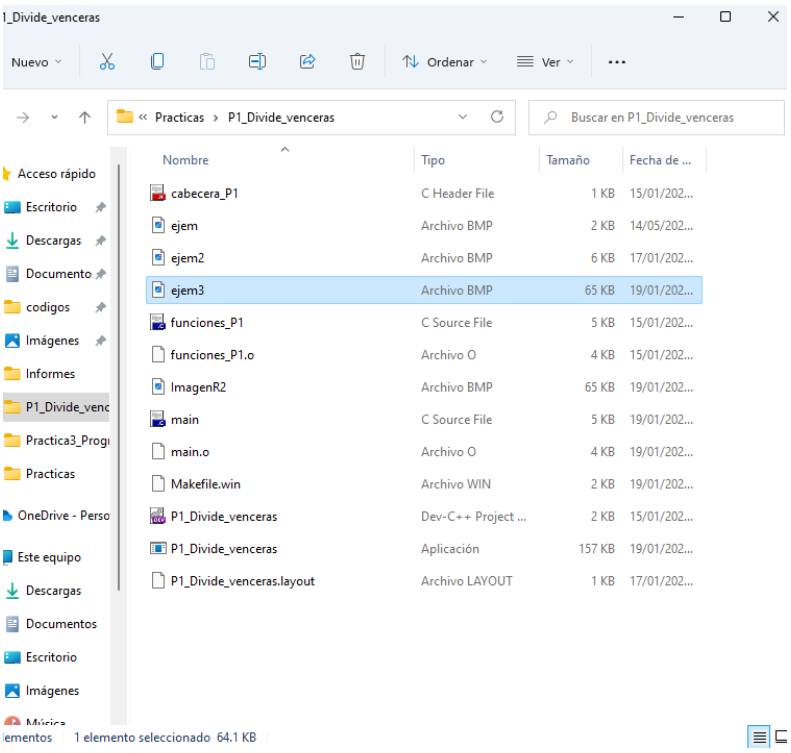
```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <math.h>

void fun(int x);

unsigned char ** asignaMemMatrizCuadrada(unsigned char**, int);
void muestraMatriz(unsigned char**, int);
unsigned char** girar(unsigned char**,int);
void copia(unsigned char **,unsigned char**, int);
void copiaDosMatrInv(unsigned char **,unsigned char**, unsigned char**, int);
void copiaDosMatr(unsigned char **,unsigned char**, unsigned char**, int);
```

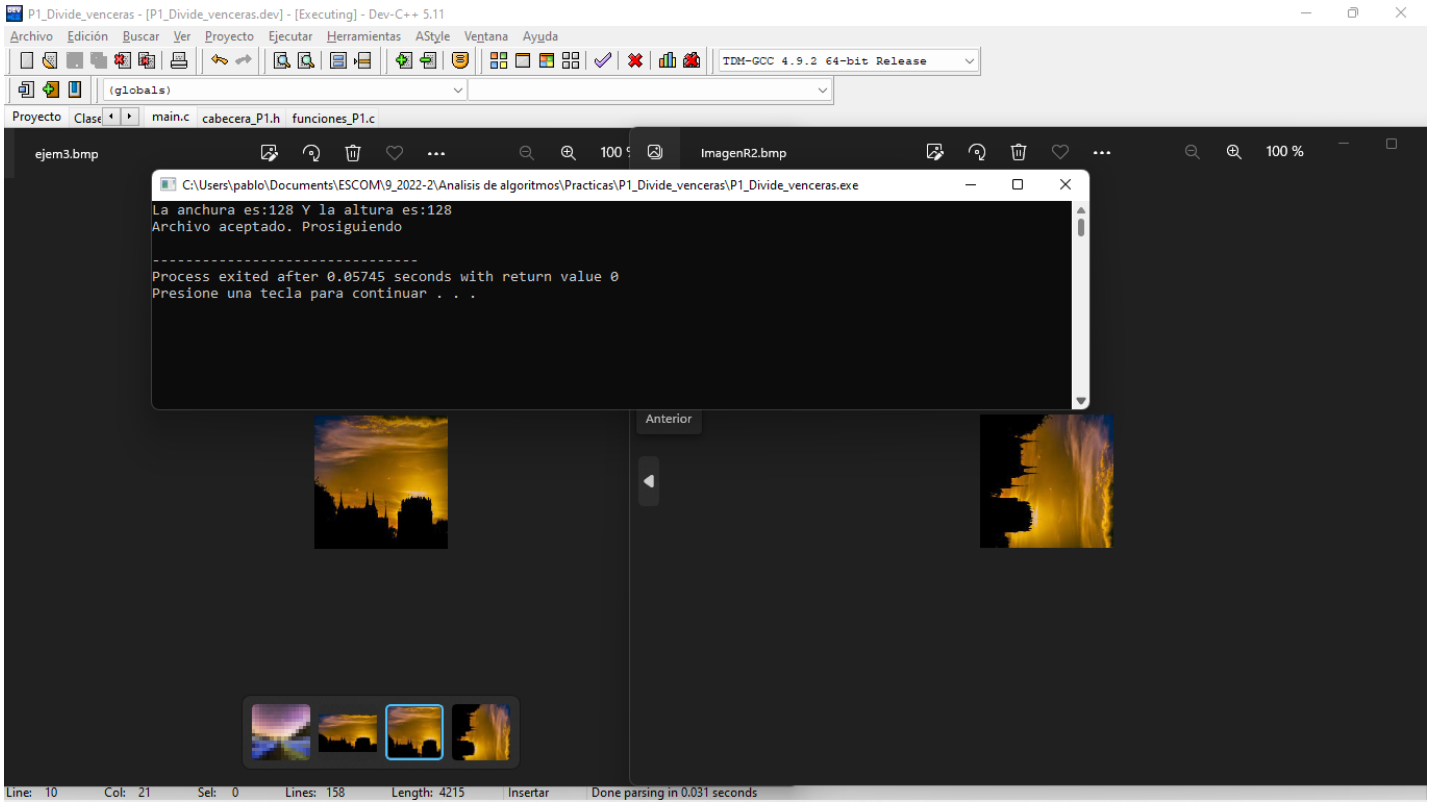
Evidencias

1. El programa lee una imagen .bmp cuadrada, donde L=2^n en caso de ser diferente nos marca un erro desde el principio diciéndonos el largo y ancho de la imagen



2. Cuando metemos una imagen de forma correcta se puede apreciar así, nos genero la nueva imagen “ImagenR2.bmp”





Referencias