Proiect de criptare si template matching

Proiectul contine 5 fisiere sursa .c (6 incluzand grayscale) si 4 fisiere de tip header.

Fisierul header bmp.h contine:

-2 struct-uri: unul pentru a retine valorile de pe cele 3 canale (R, G, B), si altul pentru a memora datele fisierului de tip bitmap (inturile fara semn width, height, char-ul ce retine informatiile din header, un pointer de tip Pixel numit values in care vom memora imaginea bmp ca un vector liniarizat de width\*height elemente, si un dublu pointer de tip Pixel numit values\_matrix, in care vom memora imaginea bmp ca o matrice[width][height];

-antetele functiilor read, write, free\_image;

In fisierul bmp.c

-functia read() de tip pointer primeste ca parametru un string ce reprezinta numele fisierului de tip bmp ale carui date dorim sa le memoram. Alocam fisierul .bmp , vectorul si matricea in mod dinamic. Calculam padding-ul cu formula din grayscale.c.

-functia write() de tip void primeste ca paramestru imaginea sub forma de struct, si un string ce reprezinta fisierul de out in care va afisa modificarile.

-functia de tip void free\_image() primeste ca parametru in care este memorat fisierul .bmp si elibereaza memoriile allocate dinamic.

Fisierul crypto.h contine antetele functiilor permute, encrypt, depermute, decrypt si chi\_test ce primesc ca parametru struct-ul cu fisierul .bmp.

In fisierul xorshift implementam XORSHIFT32, pentru a genera numere random, cu ajutorul unei chei de inceput(123456789).

In fisierul crypto.c

-In functia permute: random\_int primeste vectorul de numere random obtinute prin XORSHIFT32, sigma primeste toate valorile de la 1 la width \* height.

Declaram permuted de tip Bitmap\_Image in care vom stora imaginea cu pixelii permutati.

Aplicam algoritmul lui Durstenfeld (precum in cursul 11)

-In functia encrypt alocam dinami enc de tip Bitmap\_Image, in care vom lua imaginea permutata si o vom cripta folosind formulele din pdf-ul cu enunt, anume: plecam de la un start\_value = 987654321, pe care il xoram cu primul element din vectorul de permutari, si cu elementele rezultate in urma aplicarii algoritmului lui Durstenfeld.

In functia chi\_test calculam distributia valorilor de pe canalele R, G , B.

In fisierul temp\_matching.c avem o singura functie: temp\_matching(), ce primeste ca parametrii imaginea sursa cu cifrele scrise de mana, sabloanele cu toate cifrele, si valorile date pe 3 canale pentru o culoare.

template\_grayscale\_intensity\_average = S barat, media valorilor intensitatii grayscale din sablonul cu cifra

I\_grayscale\_intensity\_average = fI barat, media valorilor intensitatii

Sigma\_temp = Sigma sablonului, rezultat din formula.

In doua for-uri prin care parcurgem toata imaginea sursa avem:

-Sigma\_fI = Sigma sablonului, rezultat din formula.

-grayscale dintr-o portiune de 11\*15 din imaginea sursa

-functia de corelatie, corr(corr\_S\_fI). Daca aveasta este mai mare de 0.33, coloram marginile sablonului in culoarea primita.

In main avem apelurile functiilor. Aici citim(prin intermediul read() ) toate fisierele de tip .bmp.

Inainte de rulare trebuie create 3 fisiere .bmp: “encrypted.bmp”, in care afisam poza criptata, “original.bmp” in care decriptam “encrypted.bmp” si “imaginecolorata.bmp” in care facem template matching-ul.