Avadanei Antonia
Grupa 133

Pentru realizarea proiectului am folosit urmatoarele functii:

1. **void XORSHIFT32(unsigned int **aleatoare)**
   - deschide fisierul peppers.bmp pentru a citi dimensiunile imaginii, H si W
   - aloca memorie pentru vectorul transmis ca parametru, "**aleatoare**"
   - deschide fisierul cu cheia secreta pentru a citi valoarea R0
   - genereaza cele 2*H*W-1 numere aleatoare folosind algoritmul XORSHIFT32

2. void liniarizare(unsigned char **header, pixel **lin, char*nume_fisier_img)
   - deschide fisierul peppers.bmp pentru a citi dimensiunile imaginii H si W
   - aloca spatiu pentru vectorul "**header**" si citeste cele 54 de elemente ale sale
   - calculeaza padding
   - aloca memorie pentru vectorul de pixeli "**lin**"
   - citeste cele H*W elemente ale vectorului "**lin**", incepand de pe ultima linie a fisierului sursa

3.void fisier_liniarizat(char *nume_fisier_img ,char* nume_fisier_img_liniarizata)
   - deschide fisierul peppers.bmp pentru a citi dimensiunile imaginii, H si W
   - calculeaza padding
   - creeaza un nou fisier, **"imagine.liniarizata.bmp"**,care va contine imaginea in forma liniarizata
   - apeleaza functia "**liniarizare**" pentru a obtine headerul si imaginea liniarizata in vectorul **"lin**"
   - scrie in fisierul **"imagine.liniarizata.bmp"** headerul, octet cu octet
   - scrie in fisierul **"imagine.liniarizata.bmp"**vectorul "**lin**", in ordine inversa a liniilor

4.void criptare(char *nume_fisier_img,char *nume_img_criptata, char *fisier_cheie_secreta, pixel **criptata)
   - deschide fisierul cu imaginea initiala si cel cu cheia secreta
   - creeaza fisierul in care vom stoca imaginea criptata
   - apeleaza functiile **XORSHIFT32** si **liniarizare**
   - aloca memorie pentru vectorii "**permutare**", **"intermediara**" si "**criptata**"
   - genereaza permutarea aleatoare de lungime H*W in vectorul **"permutare"** folosind algoritmul lui Durstenfeld si numerele din vectorul "**aleatoare"**
   - permuta pixelii vectorului **"lin"** conform permutarii aleatoare, obtinandu-se vectorul **"intermediara"**
   - citeste R0 si SV din fisierul cu cheia secreta
   - aplica asupra fiecarui pixel din **"Intermediara"** relatia de substitutie prezentata in enuntul proiectului, obtinand vectorul **"criptata"**

$$C_k = \begin{cases} SV \oplus P'_0 \oplus R_{W*H}, & k = 0 \\ C_{k-1} \oplus P'_k \oplus R_{W*H+k}, & k \in \{1,2,\dots,W*H-1\} \end{cases}$$

- in fisierul care contine imaginea criptata scrie headerul, octet cu octet, apoi fiecare element din vectorul **“criptata”** in ordine inversa a liniilor

5.void decriptare(char *nume_fisier_criptata,char *nume_img_decriptata,char *fisier_cheie_secreta,pixel **decriptata)

- deschide fisierul cu imaginea criptata si cel cu cheia secreta
- creeaza fisierul in care vom stoca imaginea decriptata
- apeleaza functiile **XORSHIFT32** si **liniarizare**
- aloca memorie pentru vectorii “**permutare**”, **“inversa”**, **“intermediara**”, “**criptata**” si **“decriptata”**
- apeleaza functia **“criptare”**
- genereaza permutarea aleatoare de lungime H*W in vectorul **“permutare”** folosind algoritmul lui Durstenfeld si numerele din vectorul “**aleatoare”**
- genereaza pemutarea inversa in vectorul **“inversa”**
- aplica asupra fiecarui pixel din **“criptata”** relatia de substitutie prezentata in enuntul proiectului, obtinand vectorul **“intermediara”**

$$C'_k = \begin{cases} SV \oplus C_0 \oplus R_{W*H}, & k = 0 \\ C_{k-1} \oplus C_k \oplus R_{W*H+k}, & k \in \{1,2,\dots,W*H-1\} \end{cases}$$

- permuta pixelii vectorului **“intermediara”** conform permutarii inverse, obtinand vectorul **“decriptata”**
- in fisierul care contine imaginea decriptata scrie headerul, octet cu octet, apoi fiecare element din vectorul **“decriptata”** in ordine inversa a liniilor
- elibereaza memoria alocata pentru vectorii utilizati

6.void chi_squared_test(char *nume_fisier_img)

- deschide fisierul pentru care calculam valorile testului chi-patrat
- creeaza cate un vector de frecventa pentru fiecare canal de culoare, f_R, f_G, f_B si alocam spatiu pentru fiecare de 256 de elemente de tipul unsigned int
- citeste H si W imagine
- apeleaza functia “**liniarizare**”
- calculeaza frecventa estimata teoretic a oricarei valori i (0<=i<=255)
- parcurge imaginea liniarizata si incrementeaza vectorii de frecventa f_R, f_G, f_B
- calculeaza valorile testului chi-patrat pentru fiecare canal de culoare conform formulei prezentata in enuntul proiectului
- afiseaza valorile testului chi-patrat pentru fiecare canal de culoare

6. void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie)

- deschide fisierul sursa
- creeaza fisierul destinatie
- citeste W si H ale imaginii sursa
- scrie in fisierul destinatie fiecare octet al imaginii sursa
- calculeaza padding
- modifica intensitatie fiecarui pixel din imaginea destinatie conform formulei de mai jos pentru a obtine imaginea grayscale

$$R' = G' = B' = [0.299 * R + 0.587 * G + 0.114 * B]$$

7.double medie_intensitati(pixelmatrice,int i,int j)**

- calculeaza media intensitatilor grayscale a unei ferestre/sablon de dimensiune h=15*w=11 astfel:
  - initializeaza variabila medie=0;
  - parcurge fereastra/ sablonul ce are coltul din stanga sus cu coordonatele (i, j)
  - parcurge fereastra/sablonul (matrice) linie cu linie si coloana cu coloana si aduna in variabila medie suma tuturor intensitatilor grayscale (deoarece intr-o imagine grayscale cele trei canale R, G, B au valori egale, adunam in variabila medie doar canelele B)
  - imparte medie la (h*w)
- returneaza medie

8.double deviatie(pixelmatrice,int i,int j)**

- calculeaza deviatia standard a intensitatilor grayscale a unei ferestre/sablon de dimensiune h=15*w=11 astfel:
  - stocheaza, prin apelul **medie_intensitati(matrice,k,l)**, media intensitatilor grayscale din fereastra/sablon, in variabila **medie**
  - initializeaza variabila deviatie=0
  - parcurge fereastra/sablonul ce are coltul din stanga sus cu coordonatele (i, j)
  - parcurge fereastra/sablonul (matrice) linie cu linie si coloana cu coloana si aduna in variabila deviatie suma data de formula

$$\sigma_{f_I} = \sqrt{\frac{1}{n-1}\sum_{(i,j)\in f_I}\left(f_I(i,j) - \overline{f_I}\right)^2}$$

- returneaza deviatie

9. pixel alege_culoare(int cifra)

- contine un meniu switch care ne permite sa alegem o culoare in functie de numarul cifrei date ca parametru
- returneaza culoarea aleasa

10. void desenez_contur(pixel ***img_matrice,int i1,int j1,unsigned char culoareB, unsigned char culoareG,unsigned char culoareR)

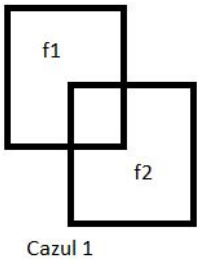
- deseneaza conturul unei ferestre ce are coltul din stanga sus cu coordonatele (i1, j1)
- parcurge linia de sus, prima culoana si ultima coloana si linia de jos si schimba valorile canalelor R, G, B cu valorile canalelor R, G, B ale culorii date ca parametru

11. int cmp(const void *corr1,const void *corr2)

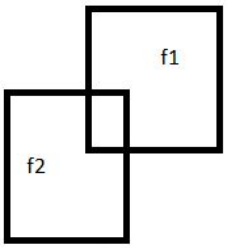
- functia comparator ce ajuta la sortarea descrescatoare a detectiilor in functie de scorul de corelatie

12.double suprapun(int indice_i_f1,int indice_j_f1,int indice_i_f2,int indice_j_f2)

- primeste ca parametrii coordonatele celor 2 ferestre si calculeaza suprapunerea, luand in calcul 2 cazuri:



Cazul 1



Cazul 2

  - calculeaza aria intersectiei, pentru fiecare caz in parte
  - calculeaza reuniune si suprapunere
  - returneaza suprapunere

13. void template_matching(char *nume_imagine, double ps, detectie **vector_detectii, int *cnt)

- deschide imaginea sursa
- citeste dimensiunile W, H
- creeaza o noua imagine, in care o stocam pe cea initiala, dar sub forma grayscale
- citim imaginea sub forma matriceala, in matricea “**img_auxiliara”**, incepand cu ultima linie
- incarcam colectia de sabloane (citite de la tastatura)
- citim h si w pentru fiecare sablon
- trecem pe rand fiecare sablon (transformat in grayscale) in forma matriceala
- calculam pe rand media intensitatilor si deviatia standard pentru fiecare sablon si le stocam in variabilele **medie_intensitati_sablon, deviatie_sablon**
- glisam, pe rand, fiecare sablon pe imaginea sursa, considerand numai pozitiile in care sablonul S incape in imaginea I
- calculam pe rand media intensitatilor si deviatia standard pentru fiecare fereastra si le stocam in variabilele **medie_intensitati_fereastra, deviatie_fereastra**
- calculam corelatia, iar daca aceasta este mai mare decat pragul ps, punem in vectorul “**vector_detectii**”, indicele de linie, indicele de coloana, corelatia, culoarea si numarul sablonului pentru detectia gasita. De asemenea setam si variabila stergere=0 pentru a ne ajuta mai tarziu la eliminarea non-maximelor.
- retinem in variabila cnt, transmisa ca parametru, numarul tuturor detectiilor gasite

14. void elimin_nonmaxime(char *nume_imagine,detectie **vector_detectii,int *cnt)

- deschide imaginea initiala (cea color)
- citeste headerul si dimensiunile H si W

- retine imaginea initiala in forma matriceala in matricea “**img_matrice”**
- sorteaza descrescator vectorul de detectii dat ca parametru, in functie de corelatie
- parcurge vectorul cu 2 for-uri pentru a calcula suprapunerea ferestrelor luate 2 cate 2
- daca gaseste o suprapunere mai mare decat 0.2 compara corelatiile celor 2 detectii si o elimina pe cea cu corelatia cea mai mica, setand variabila stergere=1 (detectiile care au stergere=1 nu vor mai fi luate in calcul la desenarea chenarelor finale)
- parcurge din nou vectorul de detectii, iar daca stergere==0, apeleaza functia de desenare a chenarelor pe imaginea **“img_matrice”**
- creeaza un nou fisier **"img_desenata_fin.bmp"** in care stocheaza imaginea finala