Proiect Programare Procedurala

2018-2019

Vilceanu Sonia Nicoleta

Grupa 144

Proiectul contine mai multe functii care alcatuiesc partea 1 (criptarea unei imagini bmp), si partea 2, (template matching intre o imagine si mai multe sabloane):

Partea 1

- Am definit 2 structuri: ***culoare*** si ***imagine***

```
struct culoare
{
   unsigned char R,G,B;
};
struct imagine
{
   unsigned int w, h, size, padding;
   struct culoare *pixeli_Img;
   unsigned char *header;
};
```

  - ***structul culoare*** contine campurile R G B in care stocheaza cei 3 octeti ai unui pixel pentru culori
  - ***structul imagine*** stocheaza in el toate informatiile despre o imagine: *size*-marimea in octeti, *h(height)*-inaltimea imaginii, *w(width)*-latimea imaginii, *header*-primii 54 de octetii in care e stocat headerul, *padding*-numarul de octeti de padding, *pixeli_Img*-vector de tip struct culoare ce contine pixelii imaginii

- functia ***xorshift32***
  unsigned int *xorshift32(unsigned int n, unsigned int seed)
  - aceasta primeste ca parametrii un seed si un numar n si genereaza n elemente pe care le returneaza sub forma unui vector. Un element este format din precedentul incepandu-se generarea de la seed.

- functia ***incarcaImg***
  struct imagine incarcaImg(const char *nume_fisier)
  - aceasta primeste ca parametru calea unui fisier imagine, pe care o liniarizeaza si a carei caracteristici le incarca intr-o variabila de tip struct imagine care va fi returnata pentru a putea fi salvata in memoria interna.

- mai intai este deschis fisierul transmis ca parametru; daca acesta nu e gasit se afiseaza un mesaj corespunzator pe ecran, iar programul se termina
- din acesta se citesc in prima faza headerul ,inaltimea, latimea, marimea in octeti si paddingul
- apoi se citeste imaginea propriu zisa, pixel cu pixel, sarindu-se peste padding la finalul fiecarei linii

- functia ***afiseaza_imagine***
  void afiseaza_imagine(const char *nume_fisier, struct imagine a)
  - primeste ca parametrii o variabila de tip struct imagine, care reprezinta imaginea sub forma liniarizata, stocata in memoria interna, si calea/numele care va fi dat imaginii pe care vrem sa o stocam in memoria externa.
  - daca exista padding se construieste un vector cu octetii de padding, cu toate valorile sale 0, care va fi afisat la final de linie, in rest pixelii se vor afisa normal, octet cu octet.
  - se elibereaza memoria ocupata de vectorul cu octetii de padding declarat dinamic.

- functia ***criptare_Img***
  void criptare_Img(const char *img_de_criptat, const char *img_criptata, const char *fisier_cheie)
  - primeste ca parametrii calea imaginii care vrem sa o criptam, calea/numele pe care dorim sa il dam fisierului in care se va pune imaginea criptata, si calea fisierului cu cheia secreta
  - se incarca, folosind functia **incarcaImg,** in forma liniarizata, in variabila *'a'* de tip *struct imagine,* imaginea pe care vrem sa o criptam.
  - generam numere aleatoare folosind functia **xorshift32**, pe care le vom pune intr-un vector 'R', mai apoi folosind primele h*w-1 elemente la generarea permutarii Durstenfeld. Aceasta permutare o salvam in vectorul 'p'. Pentru *seed* va fi folosit numarul 'r0' din fisierul cu cheia secreta.
  - intoarcem liniile imaginii pentru a putea incepe criptarea de la ultima linie, urmand ca la sfarsitul criptarii sa intoarcem imaginea la loc in pozitie normala.
  - vom crea o copie, sub forma liniarizata, a imaginii pe care vrem sa o criptam, in variabila 'a_cript'. Asupra vectorului de pixeli stocat in 'a_cript' se vor efectua toate operatiile pentru criptare.
  - se permuta pixelii conform permutarii Durstenfeld din vectorul 'p'.
  - se xoreaza pixelii dupa regula data: primul pixel cu el insusi, octetii numarului 'SV' citit din fisierul cu cheia secreta si numarul R[w*h], urmatorii formandu-se din ei insisi, precedentii lor si numerele ramase din vectorul 'R' de la indicii h*w+1 la 2*h*w-1.

- pentru a se putea lucra cu octetii unui numar, vectorul ‘R’ si numarul ‘SV’ au fost declarati intr-o *uniune* cu un camp ‘nr’ de tip unsigned int (care tine numerele propriu-zise) si un camp ‘oct[4]’ care tine impartirea lor in octeti.
- la sfarsit de stocheaza in memoria externa imaginea criptata, tinuta in variabila ‘a_cript’ sub forma liniarizata, folosindu-se functia **afiseaza_imagine.**
- se elibereaza memoria ocupata de variabilele declarate dinamic.

```
union impartireNrInOcteti
  {
    unsigned int nr;
    unsigned char oct[4];
  };
```

- functia ***decriptare_Img***
  void decriptare_Img(const char *img_criptata,const char *img_decriptata,const char *fisier_cheie)
  - aceasta este in mare parte inversul criptarii
  - se genereaza permutarea inversa lui ‘p’, ‘pinv’.
  - se intorc liniile pentru a se incepe de la ultima si se xoreaza dupa regula data, folosind o *uniune* identica, impreuna cu acelasi vector ‘R’ si valori ‘r0’ si ‘SV’ din fisierul cu cheia secreta. Xorarea se face asupra vectorului de pixeli dintr-o variabila ‘a_intermediara’, la inceput identica cu ‘a_cript’ din functia precedenta.
  - s-a declarat o variabila ‘a_decript’ asupra carei se va efectua inversarea, aceasta fiind si forma finala in care se va tine imaginea decriptata intr-o forma liniarizata.
  - liniile se rearanjeaza dupa regula elementelor din permutarea inversa, apoi fiind din nou reintoarse la normal pentru a se putea stoca corect imaginea.
  - stocarea in memoria externa se face tot cu functia **afiseaza_imagine**
  - se elibereaza memoria ocupata pe heap

- functia ***test_chi***
  void test_chi(char *imagine)
  - primeste ca parametru calea imaginii pentru care se va calcula testul chi.
  - imaginea este pusa intr-o variabila ‘a’ folosindu-se functia **incarcaImg.**
  - se calculeaza pentru fiecare canal de culoare valorile testului chi dupa formula data.
  - se elibereaza memoria ocupata pe heap.

- ***Main***
    - in functia **main** se citesc de la tastatura numele fisierului ce contine imaginea pe care dorim sa o criptam, numele pe care il dam fisierului in care sa se salveze imaginea criptata si numele fisierului cu cheia secreta.
    - se apeleaza functia **criptare_Img.**
    - se citesc de la tastatura numele imaginii criptate, numele pe care dorim sa-l dam fisierului in care va fi pusa imaginea decriptata si numele fisierului care contine cheia secreta.
    - se apeleaza functia **decriptare_Img.**
    - se apeleaza functia **test_chi** pentru imaginea initiala si pentru imaginea criptata, valorile fiind afisate pe ecran.
    - Memoria ocupata de declararile char-urilor pentru citire este eliberata

Partea 2

- Am definit o structura in plus ***fereastra***, pe care o folosim pe langa celelalte doua structuri **culoare** si **imagine**

```
struct fereastra
{
   double corelatie;
   unsigned int linie,coloana,marime;
   struct culoare culori;
};
```

- Functia ***grayscale***
  void grayscale(const char *imagineColor, const char *imagineGrayscale)
    - Are ca parametrii calea imaginii pe care dorim sa o facem alb-negru, si calea pe care dorim sa o dam fisierului binar in care va fi pusa imaginea alb-negru
    - Functia incarca intr-o variabila ‘a’ de tip struct imagine, imaginea sub forma liniarizata, folosind functia **incarcaImg.**
    - Modifica pixelii salvati in ‘a-pixeli_Img’ dupa formula data, aducand toti cei trei octeti ai unui pixel la aceeasi valoare.
    - Apeland functia **afiseaza_imagine,** se stocheaza in memoria externa imaginea grayscale.
    - se elibereaza memoria ocupata pe heap.

- functia ***template_matching***

struct fereastra *template_matching(const char *img, const char *sablon, double prag, struct imagine a)

- are ca parametrii calea unei imagini, a unui sablon, a pragului, a unei variabile de tip struct imagine
- calculeaza corelatia pentru un sablon si o imagine, salvand ferestrele care indeplinesc conditia f1.corelatie>prag intr-un 'vectorFI', f1(de tip struct fereastra) fiind fereastra care incepe de la linia i si coloana j de lungime si latime egale cu cele ale sablonului.
- se apeleaza functia **grayscale** pentru sablonul transmis ca parametru si un fisier "cifraGray.bmp" in care se va tine varianta grayscale a sablonului. Acest fisier va fi incarcat in forma liniarizata intr-o variabila de tip struct imagine, 's', si apoi va fi sters, urmand sa se lucreze cu variabila 's' pentru a reprezenta sablonul.
- Se calculeaza corelatia dupa formula data iar daca aceasta este mai mare decat pragul transmis ca parametru este adaugata in vectorul 'vectorFI' de tip struct fereastra
- -la final este returnat 'vectorFI', salvandu-se numarul de elemente al acestuia in campul marime al primului element (vectorFI [0] . marime).
- Se elibereaza campurile variabilei s din memorie.

- Functiile ***sortare*** si ***cmp***
  void sortare(struct fereastra **vectorFI)
  - Functia **sortare** primeste ca parametru un vector de tip fereastra si il sorteaza dupa conditia data de functia **cmp** (folosindu-se functia qsort din libraria stdlib.h) anume in ordine descrescatoare dupa campul corelatie.

- Functia ***suprapunere***
  double suprapunere(struct fereastra di, struct fereastra dj,struct imagine s)
  - Verifica daca doua ferestre primite ca parametru se suprapun si calculeaza suprapunerea acestora dupa formula suprapunere=intersectie/(aria f1+ aria f2-intersectie)

- Functia ***elim_nonMaxime***
  void elim_nonMaxime(struct fereastra **D, struct imagine s)
  - Primeste ca parametru un vector de ferestre 'D', pe care il sorteaza prin apelul functiei **sortare**, si un sablon sub forma liniarizata si elimina din vectorul 'D' ferestrele care au o suprapunere mai mare decat 0.2, prin apeluri ale functiei **suprapunere.**

- Functia ***coloreaza***
  void coloreaza(const char *img, struct fereastra f1, struct culoare C, struct imagine *a, struct imagine s)
    - Primeste ca parametru calea unei imagini, o fereastra, o culoare, si doi parametrii ‘a’ si ‘s’ de tip struct imagine care reprezinta sablonul curent si imaginea principala.
    - Functia coloreaza conturul unei ferestre cu culoarea C

- ***Main***
    - Se citesc de la tastatura numele imaginii pentru care vrem sa aplicam algoritmul de template matching, impreuna cu numele sabloanelor
    - Va fi creata o varianta **grayscale** a imaginii principale pe cale o vom pune sub forma liniarizata in variabila ‘a’. Folosim pentru asta un fisier temporar “imagineDeTestatGray.bmp” pe care apoi il vom sterge din memorie.
    - Apelam functia de **template_matching** pentru fiecare dintre sabloanele citite si le punem pe toate intr-un vector ‘D’, alegand si culoarea pentru fiecare sablon.
    - Apelam functia **elim_nonMaxime** pentru vectorul ‘D’ pentru a elimina toate detectiile suprapuse si parcurgem fiecare element al acestuia apeland pentru fiecare fereastra D[i] functia **coloreaza.**
    - La final se afiseaza imaginea colorata cu dreptunghiuri cu functia **afiseaza_imagine** in fisierul “imagineColorata.bmp”.
    - Se da free pentru memoria alocata dinamic.