Documentatie Proiect Programare Procedurala

Arnautu Andrei

Decembrie 2018

1 Continutul arhivei

Arhiva contine urmatoarele fisiere:

- Documentatia: documentatie.pdf
- Fisierele sursa: main.c, main_encryption.c, main_recognition.c
- Headerele: common_data.h, image_encryption.h, image_recognition.h
- Diferite resurse folosite la verificarea corectitudinii programelor: template-uri(cifra0.bmp, cifra1.bmp, ..., cifra9.bmp), imagini in format bitmap (peppers.bmp, test.bmp) si doua fisiere text (data.txt, secret_key.txt).

2 Structura proiectului

Proiectul are urmatoarea structura: toate "bucatile" de cod necesare rezolvarii cerintelor de la 1 la 10 se afla in cele 3 headere. Headerul **common_data.h** contine functiile si structurile care sunt necesare atat pentru rezolvarea cerintelor din primul modul, cat si pentru rezolvarea celor din modulul al doilea. Headerul **image_encryption.h** contine doar functiile si

structurile necesare exclusiv pentru rezolvarea cerintelor din primul modul, iar **image_recognition.h** le contine pe cele necesare pentru rezolvarea celor din cel de-al doilea modul.

In cele ce urmeaza vor fi prezentate functiile din fiecare header, incluzand utilizarea lor pentru a rezolva cele 11 cerinte propuse.

3 Headerul common_data.h

Structuri implementate:

- structura **Pixel** : un vector de 3 elemente de tip **unsigned char**, folosite pentru a stoca cele 3 intensitati (in ordinea Blue-Green-Red fiindca asa se citesc din fisierele bitmap si sunt mai usor de manipulat astfel).
- structura **BMPImage** : aceasta structura tine toate datele necesare procesarii unei imagini de tip BMP. Este formata dintr-un sir de 54 de elemente de tip **unsigned char** in care se retine headerul, un vector alocat dinamic de pixeli (fiindca tinem pixelii in forma liniarizata) si doua variabile de tip **int** pentru a retine inaltimea si latimea imaginii.

Functii implementate:

- **int** GetIndex(**int** row, **int** column, **int** width) : returneaza pozitia in vectorul liniarizat pe care se afla pixelul cu coordonatele (row, column) dintr-o imagine BMP care are latimea width.
- **int** GetPadding(**int** width): returneaza valoarea paddingului pentru fiecare linie a unei imagini BMP care are latimea width.
- **BMPImage** LoadBMPImage(**char** *file_name) : functia care rezolva **cerinta (2)**. Aceasta primeste ca parametru calea unei imagini de tip bitmap si returneaza o structura de tip **BMPImage**. Se deschide

fisierul in cauza, se stocheaza cei 54 de octeti ai headerului si se extrag mai apoi inaltimea si latimea imaginii. Mai apoi, trebuie alocati dinamic un numar de (latime * inaltime) pixeli, care vor fi cititi din fisier, avand bineinteles grija la padding. Odata ce sunt toate datele stocate intern, se inchide fisierul in care se afla imaginea BMP si se returneaza structura de tip **BMPImage**.

- **void** PrintBMPImage(**BMPImage** *image, **char** *file_name) : functia care rezolva **cerinta (3)**. Aceasta primeste ca parametri un pointer catre o imagine BMP stocata intern si calea fisierului in care trebuie salvata extern. Se afiseaza intai cei 54 de octeti care compun headerul, iar apoi pixelii care compun imaginea. De asemenea, trebuie avut grija la valoarea de padding. Daca aceasta este nenula, trebuie afisat la finalul fiecarei linii un numar de octeti egal cu valoarea de padding.

4 Headerul image_encryption.h

Functii implementate:

- **unsigned int** XorShift32(**unsigned int** former_state) : functia care rezolva **cerinta (1)**. Aceasta implementeaza generatorul de numere pseudo-aleatoare Xor-Shift 32. Primeste ca parametru numarul generat anterior sau valoarea de seed (daca numarul in cauza este primul generat) si furnizeaza urmatorul numar pseudo-aleator. Pentru a genera un sir de n astfel de numere pseudo-aleatoare, se poate apela aceasta functie de n ori, de fiecare data dand ca parametrul numarul generat anterior, iar initial valoarea de seed.

- **unsigned int*** GetRandomNumbers(**unsigned int** seed, **int** counter): functia returneaza un sir de counter numere pseudo-aleatoare folosind generatorul Xor-Shift 32 si pornind de la un seed dat. Functia este folosita pentru a genera cele 2 * W * H - 1 elemente ale sirului R, care este utilizat la criptarea si decriptarea unei imagini bitmap.

- **unsigned int*** GeneratePermutation(**int** size, **unsigned int*** random_numbers) : functia care genereaza o permutare de size elemente, folosind algoritmul lui Durstenfeld. De asemenea, functia primeste ca parametru si sirul de numere pseudo-aleatoare generate, de care este nevoie pentru implementarea algoritmului.

- **Pixel** XorPixelWithConstant(**Pixel** pixel, **unsigned int** x) : functia este folosita pentru a face operatia xor intre un pixel si un numar. Ea primeste aceste 2 valori ca si parametri si returneaza o structura de tip **Pixel** in care se afla rezultatul operatiei, dupa aplicarea formulei date.

- **Pixel** XorPixelWithPixel(**Pixel** p1, **Pixel** p2) : functia primeste ca parametri 2 pixeli si returneaza pixelul rezultat in urma aplicarii operatiei xor pe cei 2.

- **void** EncryptImage(**char*** image_file, **char*** encrypted_image_file, **char*** key_file) : functia care rezolva **cerinta (4)**. Primeste ca parametri calea unei imagini bitmap, calea fisierului unde va fi stocata imaginea bitmap in urma criptarii si calea fisierului in care se afla cheia secreta. In cadrul functiei se folosesc functiile de generare a unui sir de numere pseudo-aleatoare, de generare a unei permutari folosind algoritmul lui Durstenfeld si functiile care aplica operatia xor intre 2 pixeli sau intre un pixel si un numar. Algoritmul in sine este exact cel explicat in cadrul descrierii proiectului. Odata implementate acele 4 functii pomenite mai sus, implementarea acestei functii devine destul de usoara si clara. Se incarca imaginea initiala intr-o structura de tip **BMPImage**, se genereaza numerele pseudo-aleatoare si permutarea, iar apoi se aplica algoritmul de criptare. La final, imaginea criptata se salveaza in fisierul dat ca parametru. De asemenea, la sfarsit trebuie avut grija la dealocarea memoriei folosite pentru stocarea sirurilor de pixeli din cadrul imaginii initiale si imaginii criptate, a sirurilor de numere pseudo-aleatoare si a permutarii.

- **void** DecryptImage(**char*** encrypted_image_file, **char*** decrypted_image_file, **char*** key_file) : functia care rezolva **cerinta (5)**.

Primeste ca parametri calea unei imagini bitmap criptate, calea fisierului unde va fi stocata imaginea bitmap decriptata si calea fisierului care contine cheia secreta. Procedeul, din punct de vedere al implementarii, este foarte asemanator cu cel folosit la criptare si foloseste aceleasi functii pomenite si mai sus. Pasii de la criptare vor fi, insa, aplicati in ordine inversa de aceasta data. Si aici trebuie la final dealocata memoria folosita la stocarea sirurilor de pixeli ai imaginilor, a sirului de numere pseudo-aleatoare si a permutarii.

- **void** PrintBMPTest(**char*** file_name) : functia care rezolva **cerinta (6)**. Primeste ca parametru calea unei imagini bitmap, pe care o va incarca temporar in memoria interna. Apoi, pentru a afla rezultatele testului pentru fiecare canal de culoare, se tine un vector de frecventa a fiecarei intensitati a culorii, de la valoarea 0 la valoarea 255 si se foloseste formula data pentru a obtine valoarea testului, care se afiseaza in final la consola.

5 Headerul image_recognition.h

Structuri implementate:

- Structura **Window**: este folosita pentru a memora detectiile unui sablon intr-o imagine mai mare folosind algoritmul de Template-Matching. Daca privim pixelii ca pe o matrice de dimensiuni **height** si **width**, atunci o fereastra este unic determinata de coltul din stanga sus (variabilele **x** si **y**) si de dimensiuni (**height** si **width**). In plus, pentru a usura implementarea unor functii, cum ar fi cea in care desenam dreptunghiuri in jurul ferestrelor detectate, vom tine minte si indicele culorii cu care se va desena dreptunghiul (variabila **color_index**) si coeficientul de potrivire obtinut in urma aplicarii formulei de cross-correlation (variabila **correlation**).
- Structura **WindowArray** : ne dam seama de necesitatea acestei structuri in momentul in care trebuie implementata functia de Template-

Matching. Practic, noi dorim sa suprapunem sablonul peste imagine in toata pozitiile valide posibile si sa adaugam o fereastra la un sir de detectii in cazul in care indicele de potrivire (cross-correlation) este suficient de mare. Pe langa sirul in sine, trebuie sa retinem si numarul de elemente ale acestuia. Am ales astfel sa implementez acest lucru folosind conceptul din spatele structurii stl::vector din C++, care este foarte eficient in practica. Structura va tine minte 3 lucruri: sirul de ferestre alocat dinamic (**v**), numarul de elemente din acest sir (**size**) si numarul de elemente pentru care s-a alocat memorie in cadrul vectorului **v** (**allocated_size**). Procedeul de implementare este prezentat in cadrul functiei **Append**.

Functii implementate :

- **void** Append(**WindowArray*** window_array, **Window*** window) : functia are rolul de a adauga o fereastra noua la finalul un sir de ferestre deja existent (primeste ca parametri un pointer la sirul de ferestre si un pointer la fereastra care trebuie adaugata). Variabila **size** din cadrul structurii WindowArray va fi incrementata si se va verifica daca aceasta este mai mare decat variabila **allocated_size**, care retine pentru cate elemente este alocata memorie in sirul de ferestre. Daca nu este mai mare, deci cu alte cuvinte daca are loc in sir, atunci fereastra curenta se va adauga pe pozitia **size - 1** (sirul este indexat de la 0). Daca nu, atunci dublam variabila **allocated_size** si dublam memoria alocata pentru sirul de ferestre folosind functia **realloc()**, urmand ca abia apoi sa punem fereastra curenta pe pozitia **size - 1**. Astfel, functia **realloc()** se va folosi de foarte putine ori pe parcursul programului, rezultand intr-un timp de executie mai scazut al programului decat daca am folosi-o de fiecare data cand adaugam o fereastra noua la finalul sirului.

- **BMPImage** GetGreyscale(**const BMPImage*** image) : functia returneaza varianta greyscale a unei imagini bitmap stocata intern care este data ca si parametru printr-un pointer. Se tine in memorie o noua variabila de tip **BMPImage**, care se obtine din imaginea initiala aplicand fiecarui pixel transformarea data.

- **double** GetCrossCorrelation(**BMPImage*** image, **BMPImage*** pattern) : functia are rolul de a calcula si de a returna indicele de potrivire dintre o fereastra decupata dintr-o imagine (fereastra este stocata in variabila **image**) si un sablon (stocat in variabila **pattern**). Evident, atat fereastra cat si sablonul au aceleasi dimensiuni. In continuare se implementeaza efectiv formulele mentionate in enuntul proiectului iar la final se returneaza indicele dorit.

- **WindowArray** TemplateMatching(**BMPImage*** image, **BMPImage*** pattern, **const double** coefficient) : functia care rezolva **cerinta (7)**. Primeste ca parametri un pointer la imaginea principala, un pointer la sablon si un coeficient (adica valoarea de prag din enunt) si returneaza o variabila de tip **WindowArray**, adica sirul de ferestre din imagine peste care sablonul se potriveste cu un coeficient mai mare sau egal decat cel dat ca parametru. Primul pas este sa obtinem variantele greyscale ale imaginii si ale sablonului si apoi sa ne definim o variabila de tip **WindowArray**, care are initial **size** = 0 si **allocated_size** = 1. Urmeaza apoi sa suprapunem sablonul pe imagine in fiecare pozitie posibila si sa calculam coeficientii de potrivire. Daca dam peste o pozitie unde coeficientul de potrivire este mai mare sau egal decat cel dat ca parametru acestei functii, atunci adaugam fereastra in variabila de tip **WindowArray**, folosind functia **Append()**. La final, trebuie sa dealocam memoria folosita pentru stocarea unor siruri necesare doar in cadrul functiei si apoi sa returnam variabila de tip **WindowArray** unde am retinut ferestrele care ne intereseaza.

- **void** DrawRectangle(**BMPImage*** image, **Window** window, **Pixel** color) : functia care rezolva **cerinta (8)**. Primeste ca parametri un pointer la o imagine bitmap stocata intern, o fereastra si o culoare. Astfel, functia "coloreaza" toti pixelii aflati pe marginea ferestrei date in culoarea respectiva, parcurgand cele 4 laturi (sus, jos, stanga, dreapta).

- **int** CompareWindows(**const void*** a, **const void*** b) : este functia de comparare folosita la cerinta (9) pentru a sorta un sir de ferestre in ordinea descrescatoare a indicilor de potrivire.

- **void** SortWindows(**WindowArray*** window_array) : este functia care rezolva **cerinta (9)**. Primeste ca parametru un pointer la o variabila de tip **WindowArray**. Este, de fapt, un wrapper al apelului functiei **qsort()**. Se apeleaza functia **qsort()** dand ca parametru functia de comparare **CompareWindows()**, avand ca scop sortarea unui sir de ferestre in ordinea descrescatoare a indicilor de potrivire.

- **int** Min(**int** a, **int** b) : functie care returneaza minimul a doua numere de tip **int** date ca parametri.

- **int** Max(**int** a, **int** b) : functie care returneaza maximul a doua numere de tip **int** date ca parametri.

- **int** ComputeOverlap(**int** x0, **int** x1, **int** x2, **int** x3) : sa presupunem ca avem doua siruri de celule pe axa Ox, unul continand celulele de la x0 la x1 inclusiv si celalalt continand celulele de la x2 la x3 inclusiv. Functia returneaza numarul de celule continute de intersectia celor doua siruri si se foloseste de functiile **Min()** si **Max()**.

- **double** ComputeOverlapCoefficient(**const Window*** w1, **const Window*** w2) : functia are rolul de a calcula si de a returna suprapunerea spatiala dintre doua ferestre. Parametrii sunt 2 pointeri, cate unul pentru fiecare fereastra. Se calculeaza aria intersectiei dintre cele doua ferestre, folosindu-ne de apeluri la functia **ComputeOverlap()** si apoi se implementeaza formula data in enunt pentru calcularea coeficientului de suprapunere.

- **void** DeleteElement(**WindowArray*** window_array, **int** index) : functia are rolul de a sterge un element de pe o pozitie data dintr-un sir de ferestre.

- **void** DeleteRedundantDetections(**WindowArray*** window_array) : functia care rezolva **cerinta (10)**. Primeste ca parametru un pointer la o variabila de tip **WindowArray** si are rolul de a elimina non-maximele din sirul de ferestre. Pentru fiecare doua ferestre pentru care indicele de suprapunere depaseste o valoare fixata (0.2 in acest caz) se va sterge fereastra cu indicele de potrivire mai mic. Acest concept se implementeaza

in mod corect destul de usor cu doua instructiuni **for** si folosind apeluri la functiile **ComputeOverlapCoefficient()** si **DeleteElement()**.

6 Fisierele sursa

Folosind functiile din headerele prezentate mai sus se implementeaza destul de simplu si de direct **cerinta (11)**. In fisierul **main.c** sunt implementate toate operatiile cerute la aceasta cerinta. Totusi, este mai usor de testat intai operatiile care tin de modulul de criptare de imagini, iar mai apoi separat operatiile care tin de modulul de recunoastere de imagini. Acestea sunt puse separat in fisierele **main_encryption.c**, respectiv **main_recognition.c**.

Partea care tine de modulul de criptare de imagini se face interactionand cu programul prin intermediul consolei.

Pentru cel de-al doilea modul insa ar trebui introduse mai multe date de la tastatura, asa ca este mai usor ca datele sa fie citite dintr-un fisier "data.txt", care poate fi editat in prealabil. Datele trebuie introduse in acest fisier in urmatorul format:

- pe prima linie calea fisierului care contine imaginea bitmap peste care vom potrivi sabloanele;
- pe a doua linie numarul de sabloane pe care le vom folosi, fie el **x**;
- pe fiecare din urmatoarele **x** linii se va afla calea catre unul dintre sabloane.
- pe fiecare din urmatoarele **x** linii se vor afla cate 3 valori naturale din intervalul [0, 255] separate printr-un spatiu, care reprezinta intensitatea canalelor (Albastru, Verde, Rosu) pentru culoarea curenta. Aceste culori vor fi utilizate la desenarea dreptunghiurilor dupa operatia de Template-Matching.

In arhiva fisierul data.txt contine datele furnizate in cadrul proiectului (calea

catre imaginea principala, datele despre sabloanele si culorile folosite pentru fiecare cifra).