Documentație

Dilirici Radu

Rularea programului:

Se poate folosi proiectul în CodeBlocks “Proiect Dilirici Radu.cbp”, sau se poate copia și rula executabilul din fișierul bin în fișierul principal.

Pentru compilarea manuală, cu toate că programul este scris în limbajul C, trebue folosită comanda g++ main.c (cel puțin pe mașina mea cu Windows). Programul funcționează și dacă este compilat cu comanda gcc -std=c99 main.c, dar nu afișează corect testul chi pătrat (afișează doar 0-uri).

I. Criptarea / Decriptarea imaginii

Formatul BMP

Programul operează pe imagini cu formatul BMP. Acestea sunt stocate cu ajutorul unei structuri. Header-ul și pixelii sunt reținute in vectori alocați dinamic. Vectorul *header* reține primii 54 de biți ai imaginii. Vectorul *pixel* reține urmatorii 3 * lungime * latime biti.

```
typedef struct
{
    unsigned char *header;
    unsigned char *pixel;
}imagine;
```

Citire / Afișare

Pentru citire se folosește funcția de creare a imaginii, care aloca memorie si returneaza imaginea. De asemenea, se folosesc funcții care returneaza lungimea, lațimea imaginii. În cazul în care numărul pixelilor de pe lățime nu se divide cu 4 este calculat si folosit un padding.

XorShift32

Pentru generarea numerelor pseudo-aleatoare este folosit algoritmul XorShift32.

Xor Pixeli

Pentru a obține o distribuire mai uniforma a culorilor, pixelii sunt Xor-ați cu formula din prezentare.

$$C_k = C_{k-1} \oplus P'_k \oplus R_{W*H+k}$$

```
Rwh = u_int_to_3_u_char(random_keys[width * height + k]);

Ck1[0] = criptata.pixel[3 * (k - 1)];
Ck1[1] = criptata.pixel[3 * (k - 1) + 1];
Ck1[2] = criptata.pixel[3 * (k - 1) + 2];

Pk[0] = img_test.pixel[3 * k];
Pk[1] = img_test.pixel[3 * k + 1];
Pk[2] = img_test.pixel[3 * k + 2];

aux = XOR_pixeli(Ck1, Pk);
aux = XOR_pixeli(aux, Rwh);

criptata.pixel[3 * k] = aux[0];
criptata.pixel[3 * k + 1] = aux[1];
criptata.pixel[3 * k + 2] = aux[2];
```

Permutarea

Este generată o permutare cu ajutorul numerelor generate. Apoi este creată o imagine nouă.

```
k = permutare[i];
img_perm.pixel[3 * i] = img_test.pixel[3 * k];
img_perm.pixel[3 * i + 1] = img_test.pixel[3 * k + 1];
img_perm.pixel[3 * i + 2] = img_test.pixel[3 * k + 2];
```

Există funcții diferite care fac permutarea / Xor-area pixelilor invers (pentru decriptare).

Criptare / Decriptare

Pentru a cripta imaginea se permută pixelii imaginii, iar apoi se aplică algoritmul de Xor-are.

```
img_test = permutare_pixeli(img_test, random_keys, width, height);
img_test = XOR_culori(img_test, SV, random_keys, width, height);
```

Pentru decriptare se folosesc funcțiile inverse și în ordine inversă.

```
img_test = XOR_culori_invers(img_test, SV, random_keys, width, height);
img_test = permutare_inversa_pixeli(img_test, random_keys, width, height);
```

Testul χ^2

Pentru calcularea testului se folosesc trei vectori de frecvență (pentru fiecare canal de culoare). Acestia sunt inițializați cu 0 și cresc pe rând pe pozitia citită. (Prima oară modific în vectorul de frecvență pentru albastru, apoi pentru verde, iar apoi pentru roșu. Repet acest lucru.).

```
i = (unsigned int) aux;
if (k == 1) //Blue
    fB[i]++;
if (k == 2) //Green
    fG[i]++;
if (k == 3) //Red
    fR[i]++;
k++;
if (k == 4)
    k = 1;
```

Toate aceste funcții sunt incluse în fișierul functii_criptare.c și sunt utilizate și în partea de template matching.

II. Template Matching

Grayscale

Pentru a putea regăsi cifrele în imagine avem nevoie ca aceasta să fie alb-negru. Funcția grayscale returneaza imaginea după aplicarea formulei pixelilor.

Corelație

Funcția *corr* calculează si returnează corelația dintre o patre din imagine și un șablon. Variabila *shift* reprezintă numărul de pixeli distanță de la începutul imaginii până la colțul de început al porțiunii de imagine care este verificat.

```
pozitie = i * width_test + j + shift;
```

Template matching

Dacă corelația dintre o fereastră și un șablon trece de pragul ales, aceasta este reținută intr-un vector de ferestre. Sunt reținute corelația, coordonata din colțul stânga-jos (distanța până la pixelul respeciv, la fel ca variabila *shift*), lațimea, înaltimea și cifra pentru care s-a găsit corelația. Toate ferestrele sunt reținute intr-o structură de detecții care mai reține și numărul detecțiilor.

```
typedef struct
{
    int coord, width, height, cifra;
    double corelatie;
}fereastra;

typedef struct
{
    int nr_elem;
    fereastra *fi;
}detectii;
```

Funcția template_matching returneaza o variabilă de tip *detectii*.

Inițial, există o variabila de tip *detectii* care nu contine nicio detecție. Fiecare grup de detecții (pentru fiecare cifră) este adaugat in aceasta variabilă.

```
D_aux = template_matching(img_gray, sablon, prag, i);
adaugare_detectii(&D, D_aux);
```

După adăugarea tuturor detecțiilor în D, acestea sunt sortate. Apoi sunt eliminate detecțiile care se suprapun.

Este calculată aria de suprapunere a două căte doua ferestre, iar dacă trece de un anumit prag, cea de-a doua fereastră (cea cu corelația mai mică) este eliminată din vector, iar numărul de detecții scade cu 1.

```
double arie_suprapunere = arie(intersectie) / (arie(fi1) + arie(fi2) - arie(intersectie));
return arie_suprapunere;
```

Colorarea chenarelor

Sunt colorate mai întâi liniile de sus si jos, iar apoi cele din dreaprta si stânga. *i* și *j* reprezintă coordonata de sus / jos, respectiv stânga / dreapta, nu coordonatele x și y. Din acest motiv, in a doua parte *i* crește cu lațimea imaginii de fiecare dată (se duce în sus cu o linie). Este preluată adresa imaginii pentru a modifica direct pixelii.

Colorarea imaginii

În funcția colorare_imagine sunt inițializate toate culorile și numele fișierelor pentru cifre. Acestea sunt reținute într-o matrice cu 9 linii si 3 coloane (cele 3 canale de culori pentru fiecare dintre cele 9 culori pe care le vom folosi).

Este creată imaginea grayscale și sunt verificate cele 9 cifre pe toate pozițiile din această imagine.

După adăugarea detecțiilor, sortarea lor și eliminarea suprapunerilor se coloreaza în imaginea originală.