Alexandra Zambițchi - 133

Proiect Programare Procedurală
Documentație

Partea I - Criptare/decriptare

O structura denumita **„RGB”** reține 3 unsigned char-uri semnificând cele 3 canale de culoare ale unui pixel.

```
typedef struct
{
    unsigned char R, G, B;
}RGB;
```

O alta structura denumita **„header”** reține datele din header-ul unei imagini, dimensiunea în octeți a imaginii, lățimea și înălțimea acesteia și padding-ul.

```
typedef struct
{
    unsigned int dim_img, latime_img, inaltime_img, padding;
}header;
```

Cu ajutorul funcției **„citire_header”** se stochează datele din header-ul imaginii pe care dorim să o prelucram. Funcția primește ca parametru numele fișierului și un pointer la un element de tip „header”.

```
void citire_header(char* nume_fisier_sursa, header *date)
```

Funcția **„citire”** citește dintr-un fișier dat informația pe care o salvează sub forma unei matrice. Parametrii sunt numele fișierului din care vrem să citim informațiile, datele din header-ul acestuia și un pointer la o matrice alocate dinamic care reține pixelii din imaginea data.

```
void citire(char* nume_fisier_sursa, header date, RGB **mat)
```

Funcția **„inversare”** inversează liniile matricei citite în funcția „citire”. Deoarece primul pixel citit din imagine este cel din coltul din stânga jos, imaginea este salvată invers în matrice. Această funcție „răstoarnă” matricea astfel încât elementele să fie în ordinea în care se găsesc în imaginea inițială.

```
void inversare(RGB **mat, header date)
```

Funcția **„liniarizare”** primește ca parametrii numele fișierului din care trebuie citita informația, un pointer la un vector de tip RGB care va reține toți pixelii din imaginea dorită și datele despre header-ul fișierului. In această funcție se vor citi pixelii cu ajutorul funcției „citire”, iar matricea obținută după apelarea funcției de citire va fi transformata în vector, după apelarea funcției „inversare”.

```
void liniarizare(char* nume_fisier_sursa, RGB *imagine_liniarizata, header date)
```

Funcția **„copiere_header”** primește numele a două fișiere, unul inițial din care se vor copia datele din header în cel de-al doilea fișier.

```
void copiere_header(char* nume_fisier_destinatie, char* nume_fisier_sursa)
```

Funcția **„scriere_cript”** primește ca parametri un vector alocat dinamic de tip RGB care conține pixelii unei imagini sub forma liniarizată, numele fișierului în care vor fi scrise informațiile din vectorul liniarizat, numele imaginii sursă și datele din header-ul imaginii sursă.

```
void scriere_cript(RGB *imagine_intermediara, char* nume_fisier_destinatie, char* nume_fisier_sursa, header date)
```

Funcția **„XORSHIFT32”** generează, pornind de la un număr dat, un șir de numere aleatoare în urma unor operații pe biți. Primul element al șirului este cheia secreta, următoarele numere rezultând din operațiile pe biți ale elementelor precedente. Funcția primește numărul de început reprezentând cheia secreta a criptării/decriptării, numărul de numere care vor fi generate și un pointer la un vector alocat dinamic care va reține numerele generate.

```
void XORSHIFT32(unsigned int seed, unsigned int n, unsigned int *r)
```

Funcția **„permutare”** generează o permutare aleatoare folosind algoritmul lui Durstenfeld și numerele aleatoare generate de funcția „XORSHIFT32”. Parametrii funcției sunt numărul de elemente al permutării, șirul de numere aleatoare și un pointer la un vector care va reține elementele permutării.

```
void permutare(unsigned int n, unsigned int *r, unsigned int *perm)
```

Funcția **„inversare_linie”** primește un vector liniarizat, îl transformă în matrice, inversează matricea cu ajutorul funcției „inversare” și apoi transformă matricea înapoi în vector pentru a putea scrie în nouă imagine elementele exact cum au fost citite (începând din coltul stânga jos). Funcția primește vectorul liniarizat și dimensiunile acestuia.

```
void inversare_linie(RGB *imagine_intermediara, header date)
```

Funcția **„criptare”** primește numele imaginii finale după criptare, numele imaginii inițiale care va fi criptată și numele fișierului în care se găsește cheia secretă. În interiorul funcției se citesc datele din fișiere, se prelucrează datele conform pașilor din enunțul proiectului și apoi se scriu în imaginea nouă.

```
void criptare(char* nume_fisier_destinatie, char* nume_fisier_sursa, char * nume_fisier_cheie_secreta)
```

Funcția **„permutare_invers”** primește permutarea inițiala și numărul elementelor acesteia și un pointer la un vector care va reține elementele permutării inverse.

```
void permutare_invers(unsigned int n, unsigned int *perm, unsigned int *perminv)
```

Funcția **„decriptare”** primește numele imaginii finale după decriptare, numele imaginii inițiale care urmează a fi decriptată și numele fișierului care conține cheia secretă. În interiorul funcției se citesc datele din fișiere, se prelucrează datele din imaginea inițială conform pașilor din enunțul proiectului și apoi se scriu în imaginea nouă.

```
void decriptare(char* nume_fisier_destinatie, char* nume_fisier_sursa, char * nume_fisier_cheie_secreta)
```

Funcția **„calcul_chi_patrat”** primește numele unui fișier și calculează distribuția valorilor pixelilor într-o imagine.

```
void calcul_chi_patrat(char* nume_fisier)
```

Funcția **„chi_test”** primește numele a două imagini, una necriptată și cealaltă criptată și apelează funcția „calcul_chi_patrat” pe rand pentru fiecare fișier.

```
void chi_test(char *nume_fisier_sursa, char *nume_fisier_destinatie_cript)
```

În main, se citesc de la tastatură numele fișierelor de criptare și de decriptare, numele fișierului care conține cheia secretă și se apelează funcția „chi_test”.

Partea II - Recunoașterea de pattern-uri într-o imagine

Pentru cerințele din cea de-a doua parte am folosit din nou structura „RGB”, structura „header”, funcția „citire_header”, funcția „citire” și funcția „copiere_header”, descrise mai sus.

Structura **„detecție”** reține linia și coloana pixelului din stânga sus al unei ferestre din imaginea mare care comparată cu un șablon are corelația mai mare decât pragul stabilit. De asemenea, reține și dimensiunea ferestrei, corelația cu șablonul și culoarea cu care va trebui colorat conturul acestei ferestre.

```
typedef struct
{
    unsigned int coloana, linie, inaltime, latime;
    float corelatie;
    RGB culoare;
}detectie;
```

Funcția **„grayscale_image”** a fost preluată din materialele primite și inclusă în programul final.

```
void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie)
```

Funcția **„corelare”** primește numele unui șablon și o fereastra din imaginea inițiala, calculează corelarea după formula din enunț și returnează rezultatul calculelor.

```
float corelare(char *nume_sablon, RGB **mat)
```

Funcția **„template_matching”** primește numele imaginii inițiale, numele unui șablon, valoarea pragului, un pointer la un vector de tip „detecție”, care va reține datele despre fiecare fereastra care îndeplinește condițiile legate de corelație, și un pointer la adresa unui contor care va reține câte elemente se află în vectorul de tip „detecție”.

```
void template_matching(char *nume_fisier_sursa, char*nume_sablon, float prag, detectie *det, unsigned int *poz)
```

Cu ajutorul funcției **„qsort”** am sortat descrescător vectorul de detecții pentru a elimina ulterior suprapunerile.

```
qsort(det, poz, sizeof(detectie), cmp);
```

```
int cmp(const void *a, const void *b)
{
    if((((detectie*)a)->corelatie)>=(((detectie*)b)->corelatie))
        return -1;
    return 1;
}
```

Funcția **„colorare”** primește ca parametri un pointer la o matrice în care este salvată imaginea inițială, datele din header-ul imaginii inițiale care conține dimensiunile matricei, un pointer la vectorul care conține datele despre secvențele din imaginea inițială care se potrivesc cu șabloanele și numărul de elemente din acest vector. Funcția colorează conturul ferestrelor identificate în imaginea inițială în culoarea corespunzătoare.

```
void colorare(RGB **mat_imagine, header date_imagine, detectie *det, unsigned int poz)
```

Funcția **„scriere_pattern”** primește numele imaginii finale, numele imaginii inițiale, un pointer la vectorul care reține informații despre ferestrele din imaginea inițială care se potrivesc cu șabloanele și numărul de elemente al acestui vector. Funcția va scrie în imaginea nouă informația din imaginea inițială cu schimbările obținute după apelarea funcției „colorare”.

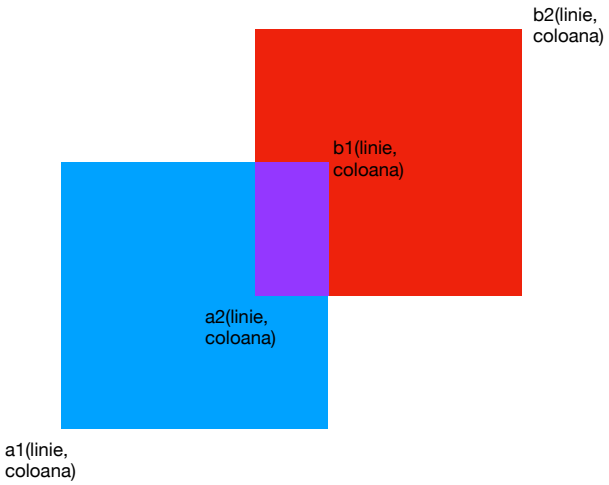
```
void scriere_pattern(char *nume_fisier_destinatie, char *nume_fisier_sursa, detectie *det, unsigned int poz)
```

Funcțiile **„min”** și **„max”** returnează minimul respectiv maximul dintre două numere primite ca parametri.

```
unsigned int min(unsigned int a, unsigned int b)
unsigned int max(unsigned int a, unsigned int b)
```

Funcția **„arie_intersectie”** furnizează aria comună a două detecții calculând indicii pixelilor din stânga jos și dreapta sus pentru fiecare detecție, știind indicii pixelului din stânga sus. Funcția returnează:
(min(b1.linie, b2.linie)-max(a1.linie, a2.linie))*(min(b1.coloana, b2.coloana)-max(a1.coloana, a2.coloana)).

```
unsigned int arie_intersectie(detectie di, detectie dj)
```



Funcția **„verif_suprapunere”** verifică dacă detecțiile primite ca parametri se suprapun sau nu. Funcția calculează indicii din dreapta jos pentru fiecare detecție, știind indicii pixelului din stânga sus. Funcția returnează 0 în cazul în care detecțiile nu se suprapun și 1 altfel.

```
int verif_suprapunere(detectie di, detectie dj)
```

Funcția **„calcul_suprapunere”** calculează suprapunerea detecțiilor din vectorul dat ca parametru. În funcție se ia cate un element și se verifică dacă se suprapune cu celelalte elemente din vector de după poziția elementului luat. Dacă se suprapune se face calculul conform formulei din enunț. Dacă rezultatul este peste limita stabilită în enunț, cea de-a doua detecție se șterge din vector.

```
void calcul_suprapunere(detectie *det, unsigned int *poz)
```

In main, se citește numele imaginii în care trebuie să se recunoască pattern-urile. Se transformă această imagine în imagine grayscale cu nume predefinit ”grayscale.bmp”. Dintr-un fișier se citesc numărul de șabloane, numele lor, numele lor după transformarea în grayscale și culorile în care trebuie colorate secventele găsite în imaginea inițială în ordinea RGB.