Documentatie Proiect la Programare Procedurala

Structurile de date folosite sunt:

* *Structura pixel* contine 3 variabile de tip unsigned char, responsabile pentru retinerea unui pixel in forma r, g, b.
* *Structura image* contine mai mult campuri, fiecare fiind folosite in modul urmator, pentru stocarea si modificarea unei imagini de tip BMP:

1. **Unsigned char \*header** este campul folosit pentru stocarea header-ului imaginii sub forma unui vector alocat dinamic de tip unsigned char.
2. **Struct pixel \*content** este folosit pentru stocarea continutului unei imagini, anume vectorul liniarizat, iar acesta va contine toti pixelii exceptand header-ul.
3. Cele patru campuri de tip unsigned int (**width, height, padding, contentSize**) sunt responsabile pentru stocarea caracteristicilor imaginii. Width si height se memoreaza direct din header, iar padding si contentSize se calculeaza la citire.

* *Uniunea binarInt* contine campurile **int x** si **unsigned char byteRepr[4]** si se foloseste pentru accesarul usoara a unui anumit octet dintr-un int. Va fi folosita pentru functia de criptare si decriptare a unei imagini.
* *Structura pairPoint* contine campurile **int x, y** (folosite pentru memorarea coordonatelor spatiale (x, y)) , **float corelation** (memoreaza corelatia a ferestrei cu coltul stanga jos in (x, y)), iar **int nrSablon** retine numarul de ordine al sablonului pentru care se afla corelatia din (x, y).
* *Structura arrayPairPoint* este o structura similara cu pairPoint, avand campurile **pairPoint\* arr** si int size. Primul se va folosi pentru a aloca un vector cu componente de tip pairPoint, iar al doilea va retine dimensiunea sa.

In continuare, voi prezenta functiile folosite la implementare si functionalitatile lor:

* **Void swap (int \*a, int \*b*)*** interschimba 2 variabile a si b.
* **Unsigned int\* xorShift(int seed, int size)** implementeaza algoritmul de generare de numere pseudo-aleatoare xorShift32, plecand de la numarul intreg **seed** si generand un numar fix de numere aleatoare (**size**). Functia returneaza un vector unsigned int\* randomSequence, care contine numerele. Aceasta functie rezolva prima cerinta din proiect.
* **Struct image loadBMP(char \*fileName)**incarca din memorie externa, de la locatia **filename**, in memoria interna o imagine BMP sub forma structurii image. Dupa deschiderea fisierului binar, se aloca memorie pentru **header** si se citeste acesta, apoi incepand de la pozitia 18 din fisier se citesc **dimensiunile imaginii(width si height)**, se calculeaza paddingul, iar la final se aloca memorie pentru **content** si se citeste octet cu octet din fisier fiecare pixel in parte. Functia returneaza imaginea citita in forma liniarizata si rezolva cea de a doua cerinta.
* **Void createBMP(char \*fileName, struct image image)**preia prin al doilea parametru o imagine **image** in forma liniarizata si creeaza in memoria externa imaginea la locatia **fileName**. Procedeul este asemanator cu cel de la functia loadBMP, doar ca in loc de citire se face scriere. Functia rezolva a treia cerinta din proiect.
* **Void criptBMP(char \*fileSource, char \*fileDestination, char \*fileKey)**preia o imagine salvata la locatia fileSource si creeaza la destinatia fileDestination o imagine criptata dupa algoritmul prezentat in enunt folosind cheia secreta de la locatia fileKey. Dupa ce se citesc cele doua numere **randomSeed** si **startingValue** regasite in fisierul text fileKey, se incarca in memoria interna **image**(imaginea initiala care trebuie criptata). Imaginea **cripImage**, ia initial valoarea imaginii initiale, urmand ca mai departe sa i se aplice modificarile aferente criptarii. Se salveaza si vectorul alocat dinamic randomSequence ce contine secventa generate de xorShift pentru **seed = randomSeed** si **size = pixelSize \* 2**, pixelSize fiind dimensiunea in pixeli a imaginii. Dupa ce se permuta pixelii din **cripImage.content** conform algoritmului lui Durstenfeld se ajunge la imaginea intermediara ce contine pixelii nemodificati, dar permutati. Mai departe, folosim uniunea binarInt, definita mai sus, pentru accesarea octetului 0, 1 si 2 al unui numar intreg (care va fi pe rand, fiecare numar din secventa aleatoare si la inceput startingValue). Folosind algoritmul de criptare prezentat in enunt, XOR-am fiecare octet al imaginii. La sfarsit, folosim **functia createBMP(fileDestination, cripImage)**, pentru a crea imaginea cripImage la locatia fileDestination. Functia rezolva a patra cerinta a proiectului.
* **Void decriptBMP(char \*fileSource, char \*fileCripted, char \*fileKey)**este functia folosita pentru decriptarea imaginii de la locatia fileCripted si creearea unei noi imagini decriptate la locatia fileSource. Se foloseste de asemenea fisierul text de la locatia fileKey. Algoritmul de decriptare este simetric celui de criptare. Diferenta este data de crearea unui vector alocat dinamic **inversPerm** ce contine permutarea inversa celei de la algoritmul de criptare. Dupa parcurgerea pasilor algoritmului, se creeaza la locatia **fileSource** imaginea decriptata folosing **struct image decriptedImage** prin functia **createBMP**. Functia rezolva a cincea cerinta a proiectului.
* **Void chiSquaredValues(char \*filePath)** calculeaza valorile chi-squared pentru fiecare canal de culoare al unei imagini memorate la locatia filePath, urmand pasii descrisi in enunt. Valorile sunt afisate pe ecran la sfarsitul functiei. Functia rezolva a sasea si ultima cerinta din **modulul de** **criptare si decriptare**.
* **Struct image grayscale(char \*fileSource)** preia o imagine de la locatia fileSource si o transforma in imagine grayscale, urmand indicatiile din enunt. Imaginea grayscale este apoi returnata de functie.
* **Float corr(int x, int y, float SPrime, float deviationS, struct image sablon, struct image image)** calculeaza corelatia ferestrei din imaginea **image** cu coltul stanga jos in pozitia (x, y) in raport cu imaginea **sablon**, toate fiind furnizate ca parametrii. Pe langa acestia, am transmis ca parametrii si valorile **SPrime** (S barat din formula prezentata in enunt) si **deviationS**(omega S din formula). Valorile sunt constante pentru fiecare sablon, astfel am decis calcularea lor o singura data pentru fiecare sablon, din motive de eficienta. Functia reproduce formula data in enunt, iar ea va fi folosita in functia de templateMatching ce urmeaza a fi descrisa mai jos.
* **Struct arrayPairPoint templateMatching(struct image image, struct image sablon, float ps)** realizeaza operatia de templateMatching pe o imaginea **image** si un sablon **sablon**, considerand pragul minim **ps**. In prima parte, functia calculeaza constantele sablonul **SPrime** si **deviations**, ca mai apoi sa calculeze corelatia fiecarei ferestre de dimensiuni **sablon.width** si **sablon.height** cu coltul stanga jos in (x, y). Pentru asta se foloseste functia prezentata mai sus, **corr**. Apoi, daca corelatia obtinuta este mai mare decat pragul **ps** se actualizeaza vectorul alocat dinamic **result**, de tip arrayPairPoint. In final, se returneaza vectorul **result**. Functia rezolva a saptea cerinta a proiectului.
* **Struct image drawBorder(struct image image, struct pairPoint f, struct pixel color, struct image sablon)** preia ca parametrii o imagine **image** in care urmeaza sa se deseneze un chenar de culoarea **color**, cu dimensiunile **sablon.width si sablon.height** cu coltul din stanga jos in pozitia **(f.x, f.y)**. Functia returneaza imaginea modificata si rezolva a opta cerinta a proiectului.
* **Struct pixel chooseColor(int i)**se foloseste pentru alegerea culorii potrivite, care depinde de numarul de ordine al sablonului. Se foloseste o instructiune decizionala **switch … case** pentru deciderea culorii potrivite.
* **Struct arrayPairPoint getCorelations(struct image grayscaleImage, const char \*sabloane, float corelationValue)** creeaza tabloul de detectii pentru colectia de sabloane din fisierul text **sabloane**, care dupa sortare, va fi folosit la eliminarea non-maximelor. Tabloul de detectii **corelationTable** este realocat de atatea ori cate sabloane avem, folosindu-ne si de functia **templateMatching** pentru a obtine un tablou de sabloane temporar, obtinut pentru imaginea grayscale si sablonul current. Sabloanele temporare se alipesc pe rand la tabloul **corelationTable**, in final obtinand tabloul de corelatii nesortat pentru un prag cu valoarea **corelationValue**. Functia returneaza tabloul **corelationTable**.
* **Int cmp(const void \*a, const void \*b)**este functia comparator folosita pentru qsort.
* **Int min(int a, int b) si int max(int a, int b)**returneaza minimul, respective maximul dintre 2 numere si vor fi folosite pentru gasirea suprapunerilor dintre ferestre.
* **Void deleteElement(struct pairPoint\*\* array, int\* size, int j)** primeste ca parametru un tablou alocat dinamic de tip pairPoint\*, dimensiunea sa si un indice j, iar functia elimina valoarea de pe pozitia j.
* **Int overlap(struct pairPoint p1, struct pairPoint p2, struct image sablon)** primeste ca parametrii 2 puncte care reprezinta colturile stanga jos ale dreptunghiurilor de dimensiune **sablon.width** si **sablon.height** si decide daca cele doua dreptunghiuri se suprapun. Pentru verificare, se calculeaza colturile stanga sus si dreapta jos si se compara pozitiile acestora.
* **Int overlapingArea(struct pairPoint p1, struct pairPoint p2, struct image sablon)** calculeaza suprafata comuna a doua dreptunghiuri care se suprapun folosind colturile stanga jos si dreapta sus.
* **Struct arrayPairPoint nonMaxElimination(struct arrayPairPoint corelationTable, struct image sablon)** implementeaza algoritmul de eliminare a non-maximelor. Se folosesc functiile explicate mai sus pentru a decide daca doua ferestre se suprapun, si daca da, pentru a calcula suprafata comuna a acestora. Daca valoarea **suprapunere**, care se calculeaza dupa formula din enunt, este mai mare ca 0.2, elementul de pe pozitia j se sterge. Functia returneaza tabloul **corelationTable** obtinut dupa stergerile facute si rezolva cerinta 10 din proiect.
* **Void drawFinalImage(struct originalImage, struct image sablon, struct arrayPairPoint corelationTable, const char\* finalImageDestination)** coloreaza chenarele pentru toate sabloanele, folosind functia **chooseColor** si **drawBorder**. La sfarsit, creeaza imaginea finala la locatia **finalImageDestination**.

In main, se rezolva cerinta 11 punand cap la cap toate functiile descrise mai sus.