



Tema 3. Image processing

- Responsabili:
 - Ștefan Laurențiu
 - Lemeni Ioan Codruț
 - Marius Iftimie
- Data publicării: **13.12.2019**
- Deadline: **13.01.2020 23:55**



Atentie! deadline-ul soft coincide cu cel hard! Prin urmare, nu se vor mai accepta submitii dupa data de 13.01.2020!

Actualizări:

- **13.12.2019** adaugat enunt;
- **15.12.2019** actualizat enunt cu anumire precizari legate de bonus (numele executabilului);
- **15.12.2019** adaugat checker;
- **16.12.2019** actualizat teste filtre de pooling sa aiba doar dimensiune impara;
- **16.12.2019** actualizat enunt cu precizarea ca doar la task-ul 2 headerele imaginii trebuie modificate explicit;
- **20.12.2019** actualizat enunt la task-ul 2, acum sunt specificate explicit ce campuri din headere trebuie modificate

Scopul temei:

- utilizarea structurilor;
- lucrul cu fisiere binare;
- alocarea dinamica a memoriei;
- abordarea cu pasi mici a unei probleme mai complexe;
- observarea unor tehnici de baza care se folosesc in image processing.

Introducere

Procesarea de imagini se refera la aplicarea unor algoritmi specifici pe continutul unei imagini pentru a obtine anumite

Resurse generale


- Regulament: seria CA
- Regulament: seria CB/CD
- Calendar
- Catalog laborator
- Debugging
- Coding style - CA
- Configuratie vmchecker - CA
- Test practic - CA
- Checker laborator CB/CD

Cursuri

Continutul Tematic

Laboratoare

01. Unelte de programare
02. Tipuri de date. Operatori.
03. Instrucțiunile limbajului C
04. Funcții
05. Tablouri. Particularizare - vectori
06. Matrice. Operații cu matrice
07. Optimizarea programelor folosind operații pe biți

efecte (blur, sharpening, etc.) sau rezultate (face detection/recognition, etc.). In aceasta tema vom lucra cu unul dintre cele mai simple formate de imagini si anume formatul  BMP.

O imagine BMP are următoarea structură:

- un **File Header** care are următoarele câmpuri:
 1. **signature** – 2 octeți - literele 'B' și 'M' în ASCII;
 2. **file size** – 4 octeți – dimensiunea întregului fișier;
 3. **reserved** – 4 octeți – nefolosit;
 4. **offset** – 4 octeți – offsetul de la începutul fișierului până la începutului bitmap-ului, adica al matricii de pixeli.
- un **Info Header** care poate avea structuri diferite, însă noi vom lucra cu cel care se numește **BITMAPINFOHEADER**. Are următoarele câmpuri:
 1. **size** – 4 octeți – dimensiunea Info Header-ului, care are o valoare fixă, 40;
 2. **width** – 4 octeți – lățimea matricii de pixeli (numărul de coloane);
 3. **height** – 4 octeți – înălțimea matricii de pixeli (numărul de rânduri);
 4. **planes** – 2 octeți – setat la valoarea fixă 1;
 5. **bit count** – 2 octeți – numărul de biți per pixel. În cazul nostru va avea mereu valoarea 24, adică reprezentăm fiecare pixel pe 3 octeți, adică cele 3 canale, RGB;
 6. **compression** – 4 octeți – tipul de compresie. Acest câmp va fi 0;
 7. **image size** – 4 octeți – se referă la dimensiunea matricii de pixeli, inclusiv padding-ul adăugat (vedeți mai jos);
 8. **x pixels per meter** – 4 octeți – se referă la rezoluția de printare. Pentru a simplifica puțin tema, veți seta acest câmp pe 0. Nu o să printăm imaginile.
 9. **y pixels per meter** – la fel ca mai sus;
 10. **colors used** – numărul de culori din paleta de culori. Aceasta este o secțiune care va lipsi din imaginile noastre BMP, deoarece ea se află în general imediat după **Info Header** însă doar pentru imaginile care au câmpul **bit count** mai mic sau egal cu 8. Prin urmare, câmpul va fi setat pe 0;
 11. **colors important** – numărul de culori importante. Va fi, de asemenea, setat pe 0, ceea ce înseamnă că toate culorile sunt importante.
- **BitMap**-ul, care este matricea de pixeli și care ocupă cea mai mare parte din fișier. Trei lucruri trebuie menționate despre aceasta:
 1. pixelii propriu-ziși se află într-o matrice de dimensiune **height x width**, însă ea poate avea o

- 08. Pointeri. Abordarea lucrului cu tablouri folosind pointeri
- 09. Alocarea dinamică a memoriei. Aplicații folosind tablouri și matrice
- 10. Prelucrarea șirurilor de caractere. Funcții. Aplicații
- 11. Structuri. Uniuni. Aplicație: Matrice rare
- 12. Operații cu fișiere. Aplicații folosind fișiere.
- 13. Parametrii liniei de comandă. Preprocesorul. Funcții cu număr variabil de parametri
- 14. Recapitulare

Teme de casă (general)

- Indicații generale

Teme de casă: seria CA

Teme CB/CD

- Tema 1
- Tema 2
- Tema 3
- Tema 4

Table of Contents

- Tema 3. Image processing
 - Introducere
 - Cerinte
 - Task 1: Black&White (10p):
 - Task 2: No-

dimensiune mai mare de atât din cauza **paddingului**. Acest padding este adăugat la sfârșitul fiecărei linii astfel încat fiecare linie să înceapă de la o adresă (offset față de începutul fișierului) multiplu de 4. Mare atenție la citire, pentru că acest padding trebuie **ignorat** (fseek). De asemenea, la scriere va trebui să puneți **explicit** valoarea 0 pe toți octeții de padding.

2. este **răsturnată**, ceea ce înseamnă că prima linie în matrice conține de fapt pixelii din extremitatea de jos a imaginii. Vedeti exemplul de mai jos;
3. canalele pentru fiecare pixel sunt în ordinea BGR (**B**lue **G**reen **R**ed).

Header-ele pe care le puteți folosi în implementare se află în scheletul de cod asociat temei.



Următiți cu foarte mare atenție exemplul de [aici](#) și încercați să înțelegeți cum este reprezentată o imagine BMP **înainte** de a începe implementarea. Dacă e ceva neclar, puteți întreba oricând pe forum.



Puteți folosi utilitare precum **xxd** sau **hexdump** pentru a putea vizualiza conținutul unui fișier binar (în cazul nostru, a unei imagini bmp). Vedeti și răspunsurile de [aici](#).

Crop (20p):

- Task 3: Convolutional Layers (20p):
- Task 4: Min/Max Pooling Layer (10p)
- Task 5: Clustering (30p)
- Task 6: Clean Valgrind BONUS (20p)
- Format input
- Format output
- Resurse și checker-ul local
- Trimitere tema
- Restricții și precizări
 - Listă depunctă

Cerinte

Veti avea de rezolvat 5 task-uri obligatorii și un task bonus.

Task 1: Black&White (10p):

Acest task presupune transformarea unei imagini color într-o imagine alb-negru.

Procedeu prin care o imagine color se transforma într-o imagine alb-negru este următorul: **fiecare** pixel din imaginea color, fie acesta (R, G, B), va deveni (X, X, X) unde $X = \lfloor (R + G + B) / 3 \rfloor$ unde prin $\lfloor \rfloor$ se înțelege **parte întreagă** (inferioară). De exemplu pixelul (3, 2, 2) va deveni (2, 2, 2), iar pixelul (10, 20, 30) va deveni (20, 20, 20).

Dacă numele imaginii este **<nume>.bmp** atunci imaginea

alb-negru se va scrie in fisierul **<nume>_black_white.bmp** (de exemplu, daca prima linie din fisierul input.txt este image.bmp atunci imaginea alb negru se va scrie in fisierul image_black_white.bmp).

Exemplu concret: Daca imaginea color arata astfel:



Dupa transformare ar trebui sa arate astfel:



Task 2: No-Crop (20p):

Cu siguranta vi s-a intamplat cel putin o data sa nu puteti posta o poza pe o retea de socializare (ex. Instagram) din motiv ca poza respectiva nu este patratica si aceasta sa fie decupata, pierzand informatii utile din imaginea initiala. Ne dorim sa implementam functionalitatea unei aplicatii care rezolva aceasta problema, operatia asociata mai fiind numita de no-crop. Acest task presupune bordarea unei imagini in scopul obtinerii variantei patratiche a acesteia.

Procedeu prin care se transforma o imagine oarecare intr-o imagine patratica este:

- Se determina maximul intre latimea si inaltimea imaginii
- Dimensiunea ce nu este maxima trebuie "completata" in mod simetric cu **pixeli albi (valoarea (255,255,255))** la margini (stanga si dreapta la completarea coloanelor, respectiv sus si jos la completarea liniilor) astfel incat dimensiunea mai mica sa devina egala cu cealalta dimensiune
- In cazul in care diferenta intre dimensiunile imaginii este impara, atunci prima zona de completare va lua partea intreaga din diferenta (stanga sau sus), iar a doua zona va lua partea intreaga din diferenta + 1

Sa presupunem ca avem urmatoarea imagine:

```
(1 1 1) (1 1 1) (1 1 1) (1 1 1) (1 1 1)
(1 1 1) (1 1 1) (1 1 1) (1 1 1) (1 1 1)
```

Observam ca inaltimea este egala cu 2, iar latimea este egala cu 5. Prin urmare, inaltimea imaginii trebuie completata astfel incat aceasta sa devina egala cu latimea (5). Diferenta intre cele doua dimensiuni este 3, asa ca imaginea rezultat va arata astfel:

```
(255 255 255) (255 255 255) (255 255 255) (255 255
255) (255 255 255)
(1 1 1) (1 1 1) (1 1 1) (1 1 1)
(1 1 1)
(1 1 1) (1 1 1) (1 1 1) (1 1 1)
(1 1 1)
(255 255 255) (255 255 255) (255 255 255) (255 255
255) (255 255 255)
(255 255 255) (255 255 255) (255 255 255) (255 255
255) (255 255 255)
```

Daca numele imaginii este **<nume>.bmp** atunci imaginea rezultata patratica se va scrie in fisierul **<nume>_nocrop.bmp** (de exemplu, daca prima linie din fisierul de input este image.bmp atunci imaginea rezultata se va scrie in fisierul image_nocrop.bmp).

Exemplu concret: Daca imaginea initiala arata astfel:



Atunci noua imagine ar trebui sa arate astfel:






Frame-ul atasat imaginii de mai sus are doar scopul de a evidentia marginile imaginii intrucat fundalul paginii de wiki este tot alb.



La acest task veti avea de adus modificari si in headerele imaginii rezultat, dar numai in campurile **width** si **height** din Info Header!

Task 3: Convolutional Layers (20p):

In cadrul acestui task va trebui sa aplicati anumite  filtre pe o imagine. Un filtru convolutional este o matrice care este aplicata fiecarui pixel din imagine pentru a obtine noul pixel. In continuare o sa vedem ce inseamna sa aplici o matrice (filtru de 3x3) unui pixel. Fie (B_{22}, G_{22}, R_{22}) un pixel din imagine care are urmatoorii vecini:

$$\begin{bmatrix} (B_{11}, G_{11}, R_{11}) & (B_{12}, G_{12}, R_{12}) & (B_{13}, G_{13}, R_{13}) \\ (B_{21}, G_{21}, R_{21}) & (B_{22}, G_{22}, R_{22}) & (B_{23}, G_{23}, R_{23}) \\ (B_{31}, G_{31}, R_{31}) & (B_{32}, G_{32}, R_{32}) & (B_{33}, G_{33}, R_{33}) \end{bmatrix}$$

si fie filtrul:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Atunci, in imaginea rezultata in urma aplicarii filtrului **A**, pixelul (B_{22}, G_{22}, R_{22}) va fi inlocuit cu $(B'_{22}, G'_{22}, R'_{22})$ unde:

$$B'_{22} = \sum_{i=1}^3 \sum_{j=1}^3 B_{ij} a_{ij}$$

$$G'_{22} = \sum_{i=1}^3 \sum_{j=1}^3 G_{ij} a_{ij}$$

$$R'_{22} = \sum_{i=1}^3 \sum_{j=1}^3 R_{ij} a_{ij}$$



Daca in urma calculului uneia dintre cele 3 valori de mai sus, suma rezultata este < 0 atunci valoarea se va seta la 0. Asemnator, daca suma este > 255 atunci componenta corespunzatoare din pixel se va seta la 255.



Daca un pixel are vecini in afara imaginii, la calculul sumei acesti vecini vor avea valoarea (0, 0, 0).

Sa presupunem ca avem urmatoarea imagine:

(2 2 2)	(3 3 3)	(0 0 0)	(0 0 0)
(10 10 10)	(1 1 1)	(0 0 0)	(0 0 0)
(0 0 0)	(0 0 0)	(50 50 5)	(0 0 0)
(0 0 0)	(0 0 0)	(240 0 240)	(0 0 0)



Primul pixel (cel de pe prima linie si de pe prima coloana) trebuie sa fie pixelul din coltul stanga sus al imaginii (atunci cand este afisata pe ecran)! Atentie la faptul ca imaginea se citeste rasturnata din fisierul BMP. La aplicarea filtrelor trebuie sa considerati dispunerea de mai sus (nu cea din fisier). Cu alte cuvinte, liniile sunt dispuse crescator de sus in jos, prima linie fiind si prima linie afisata pe ecran, iar coloanele sunt dispuse crescator de stanga la dreapta.

Pentru imaginea de mai sus, fisierul BMP ar arata astfel (ignorand cele 2 headere):

(0 0 0)	(0 0 0)	(240 0 240)	(0 0 0)	No
padding				
(0 0 0)	(0 0 0)	(50 50 5)	(0 0 0)	No
padding				
(10 10 10)	(1 1 1)	(0 0 0)	(0 0 0)	No
padding				
(2 2 2)	(3 3 3)	(0 0 0)	(0 0 0)	No
padding				

Pe imaginea de mai sus dorim sa aplicam urmatorul filtru:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Imaginea rezultata va fi:

(15 15 15)	(6 6 6)	(3 3 3)	(0 0 0)
(13 13 13)	(14 14 14)	(51 51 6)	(0 0 0)
(10 10 10)	(51 51 6)	(255 50 245)	(50 50 5)

```
(0 0 0)      (240 0 240) (255 50 245) (240 0 240)
```

Pentru rezolvarea acestui task va trebui sa aplicati un filtru asupra imaginii citite la taskurile anterioare.

Daca numele imaginii este **<nume>.bmp** atunci imaginea rezultata dupa aplicarea filtrului se va scrie in fisierul **<nume>_filter.bmp** (de exemplu, daca prima linie din fisierul input.txt este image.bmp atunci imaginea rezultata se va scrie in fisierul image_filter.bmp).

Exemplu concret: Daca imaginea initiala arata astfel:



Iar filtrul pe care dorim sa il aplicam este:


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Atunci noua imagine ar trebui sa arate astfel:



Filtrele de convolutie vor avea mereu dimensiune impara!

Task 4: Min/Max Pooling Layer (10p)

In cadrul acestui task va trebui sa aplicati anumite  filtre de pooling pe o imagine, si anume Max Pooling Layer respectiv

Min Pooling Layer. Un filtru de pooling este o operatie care este aplicata fiecarui pixel din imagine pentru a obtine un nou pixel. In continuare o sa vedem ce inseamna sa aplici o operatie de min/max pooling unui pixel. Fie (B_{22}, G_{22}, R_{22}) un pixel din imagine care are urmatoorii vecini:

$$\begin{bmatrix} (B_{11}, G_{11}, R_{11}) & (B_{12}, G_{12}, R_{12}) & (B_{13}, G_{13}, R_{13}) \\ (B_{21}, G_{21}, R_{21}) & (B_{22}, G_{22}, R_{22}) & (B_{23}, G_{23}, R_{23}) \\ (B_{31}, G_{31}, R_{31}) & (B_{32}, G_{32}, R_{32}) & (B_{33}, G_{33}, R_{33}) \end{bmatrix}$$

Atunci, in imaginea rezultata in urma aplicarii filtrului de pooling **max** de dimensiune **3**, pixelul (R_{22}, G_{22}, B_{22}) va fi inlocuit cu $(B'_{22}, G'_{22}, R'_{22})$ unde:

$$\begin{aligned} B'_{22} &= \max(B_{ij} \text{ a } ij), \quad i = 1..3, j = 1..3 \\ G'_{22} &= \max(G_{ij} \text{ a } ij), \quad i = 1..3, j = 1..3 \\ R'_{22} &= \max(R_{ij} \text{ a } ij), \quad i = 1..3, j = 1..3 \end{aligned}$$

Analog se aplica si pentru filtrul de pooling **min**.



In functie de dimensiunea specificata pentru filtru, se va considera pentru fiecare element matricea corespunzatoare de dimensiune **k**, unde **k** este dimensiunea filtrului de pooling.



Daca un pixel are vecini in afara imaginii, la calculul minimului/maximului acesti vecini vor avea valoarea (0, 0, 0).

Sa presupunem ca avem urmatoarea imagine:

(2 2 2)	(3 3 3)	(0 0 0)	(0 0 0)
(10 10 10)	(1 1 1)	(7 7 7)	(0 0 0)
(12 12 12)	(0 0 0)	(50 50 5)	(0 0 0)
(0 0 0)	(0 0 0)	(240 0 240)	(0 0 0)



Primul pixel (cel de pe prima linie si de pe prima coloana) trebuie sa fie pixelul din coltul stanga sus al imaginii (atunci cand este afisata pe ecran)! Atentie la faptul ca imaginea se citeste rasturnata din fisierul BMP. La aplicarea filtrelor trebuie sa considerati dispunerea de mai sus (nu cea din fisier). Cu alte cuvinte, liniile sunt dispuse crescator de sus in jos,

prima linie fiind si prima linie afisata pe ecran, iar coloanele sunt dispuse crescator de stanga la dreapta.

Pentru imaginea de mai sus, fisierul BMP ar arata astfel (ignorand cele 2 headere):

(0 0 0)	(0 0 0)	(240 0 240)	(0 0 0)	No
padding				
(12 12 12)	(0 0 0)	(50 50 5)	(0 0 0)	No
padding				
(10 10 10)	(1 1 1)	(7 7 7)	(0 0 0)	No
padding				
(2 2 2)	(3 3 3)	(0 0 0)	(0 0 0)	No
padding				

Pe imaginea de mai sus dorim sa aplicam filtrul **max** de pooling de dimensiune 3:

Imaginea rezultata va fi:

(12 12 12)	(240 50 240)	(240 50 240)	(240 50 240)
(12 12 12)	(240 50 240)	(250 50 240)	(240 50 240)
(12 12 12)	(50 50 12)	(50 50 7)	(50 50 7)
(10 10 10)	(10 10 10)	(7 7 7)	(7 7 7)

Pentru rezolvarea acestui task va trebui sa aplicati un filtru de pooling (diferentiat prin **m** pentru filtru de min, respectiv **M** pentru filtru de max) asupra imaginii citite la taskurile anterioare.

Daca numele imaginii este **<nume>.bmp** atunci imaginea rezultata dupa aplicarea filtrului se va scrie in fisierul **<nume>_pooling.bmp** (de exemplu, daca prima linie din fisierul input.txt este image.bmp atunci imaginea rezultata se va scrie in fisierul image_pooling.bmp).

Exemplu concret: Daca imaginea initiala arata astfel:



Iar filtrul pe care vrem sa il aplicam este unul de **max pooling** de dimensiune 5, imaginea rezultata va fi:



Filtrele de pooling vor avea mereu dimensiune impara!

Task 5: Clustering (30p)

In cadrul acestui task, ne propunem sa grupam (clusterizam) pixelii din cadrul imaginii noastre in zone asemanatoare de pixeli cu scopul de a evidentia aceste zone vizual sau pentru a transforma imaginea initiala intr-o imagine cat mai artificiala/simplista (o sa vedem in exemplul concret cum arata o imagine clusterizata).

Pentru a stii cum incadram doi pixeli oarecare intr-o zona, operatia de clustering va primi ca input o valoare de **threshold** (prag) si o va folosi dupa cum este descris si in algoritmul de mai jos asociat operatiei.

Algoritmul de clustering pe care o sa il implementati (si descrie formal ceea ce a fost descris mai sus) este urmatorul:

1) Se alege un pixel din imagine care nu a fost inclus in nicio zona. Fie acest pixel (R, G, B) . Daca toti pixelii au fost inclusi intr-o zona atunci algoritmul s-a terminat;

2) Pornind de la acest pixel se determina zona de pixeli care satisface simultan urmatoarele conditii:

- Pixelul ales la punctul **1)** face parte din aceasta zona;
- Oricare ar fi un pixel din zona (R', G', B') trebuie ca $|R - R'| + |G - G'| + |B - B'| \leq \text{threshold}$ unde threshold este o valoare data ca input algoritmului. De asemenea pixelul (R', G', B') nu trebuie sa fi fost deja inclus intr-o alta zona. Daca (R', G', B') era deja inclus intr-o alta zona, chiar daca s-ar indeplinii conditia ca suma modulelor sa fie mai mica decat un threshold atunci pixelul (R', G', B') nu va fi inclus in zona curenta;
- Zona gasita este maximala. Cu alte cuvinte, nu mai exista nici un alt pixel care sa fie vecin al zonei si sa indeplineasca conditiile de la punctul anterior.

3) Se determina suma valorilor pixelilor din zona curenta cat si numarul acestora (pentru a aproxima culoarea pe care o sa o aiba zona curenta).

N = numarul de pixeli din zona curenta

$$\text{sum}_R = \sum_{i=1}^N R_i$$

$$\text{sum}_G = \sum_{i=1}^N G_i$$

$$\text{sum}_B = \sum_{i=1}^N B_i$$

(R_i, G_i, B_i) reprezinta valoarea pixelului i din zona curenta;

4) Fiecare pixel (R', G', B') din zona determinata va avea ca valoare noua media aritmetica a valorilor pixelilor din zona (culoarea zonei), adica $(\text{sum}_R/N, \text{sum}_G/N, \text{sum}_B/N)$;

5) Dupa deteminarea noii zone se reia algoritmul incepand cu pasul **1**).



Modul in care alegem pixelul la pasul **1)** poate inflenta calitatea algoritmului. Pentru aceasta tema vom aplica urmatoarea metoda de selectie: daca exista mai multi pixeli care nu au fost deja inclusi intr-o zona, se va alege acel pixel care se afla pe linia mai mica, iar in caz de egalitate (pixeli pe aceeasi linie) se va alege cel care se afla pe coloana mai mica.
Atentie! Prima linie este linia cea mai de sus din imagine (adica cea mai de jos din fisier) iar prima coloana este coloana cea mai la stanga in imagine.

Sa presupunem ca avem urmatoarea imagine (rasucita deja si in ordinea (R, G, B)) si threshold = 10:

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30 29
31)
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (22 20
23)
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (22 21
23)
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (22 20
23)
```

Initial niciun pixel nu este inclus in vreo zona. Se alege pixelul din coltul stanga-sus de valoare (10 10 10) deoarece acesta nu se afla deja in nicio zona. Se determina zona maximala din care acesta face parte:

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30
29 31)
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (22
20 23)
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (22
21 23)
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (22
20 23)
```

Calculam numarul de elemente si sumele asociate zonei curente:

$N = 12$

$sum_R = 130$, $sum_G = 130$, $sum_B = 129$

Prin urmare, fiecare pixel din zona rosie va fi inlocuit cu $(sum_R/N, sum_G/N, sum_B/N)$, deci (10, 10, 10) .

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30
29 31)
(10 10 10) (10 10 10) (10 10 10) (30 28 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 33 30) (22
21 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 31) (22
20 23)
```

Alegem din nou un pixel care nu a fost inclus intr-o zona conform algoritmului de selectie. Acest pixel este cel de pe linia 1 si coloana 4 (indexarea incepe la 1) de valoare (30 30 30). Se determina zona maximala din care acesta face parte (colorata cu verde):

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30
29 31)
(10 10 10) (10 10 10) (10 10 10) (30 28 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 33 30) (22
21 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 31) (22
20 23)
```

Calculam numarul de elemente si sumele asociate zonei curente:

$N = 5$

$sum_R = 150$, $sum_G = 150$, $sum_B = 152$

Prin urmare, fiecare pixel din zona verde va fi inlocuit cu $(sum_R/N, sum_G/N, sum_B/N)$, deci (30, 30, 30) .

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30
30 30)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
21 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
```

Alegem din nou un pixel care nu a fost inclus intr-o zona conform algoritmului de selectie. Acest pixel este cel de pe linia 2 si coloana 5 (indexarea incepe la 1) de valoare (12 10 13). Se determina zona maximala din care acesta face parte (colorata cu albastru):

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30
30 30)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
21 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
```

Calculam numarul de elemente si sumele asociate zonei curente:

$N = 3$

$sum_R = 66$, $sum_G = 61$, $sum_B = 69$

Prin urmare, fiecare pixel din zona albastra va fi inlocuit cu $(sum_R/N, sum_G/N, sum_B/N)$, deci (22, 20, 23) .

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30
30 30)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (22
20 23)
```

Am determinat astfel 3 zone similare si putem spune ca am clusterizat imaginea. Aceasta imagine este cea care va fi

afisata in fisierul .bmp corespunzator acestui task.

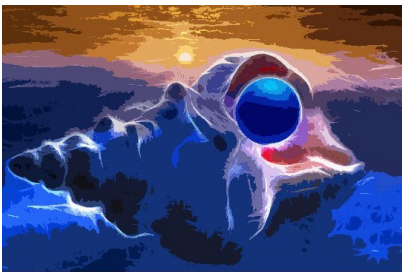
Pentru rezolvarea acestui task va trebui sa aplicati operatia de clusterizare asupra imaginii citite la taskurile anterioare.

Daca numele imaginii este **<nume>.bmp** atunci imaginea rezultata dupa aplicarea filtrului se va scrie in fisierul **<nume>_clustered.bmp** (de exemplu, daca prima linie din fisierul input.txt este image.bmp atunci imaginea rezultata se va scrie in fisierul image_clustered.bmp).


Exemplu concret: Daca imaginea initiala arata astfel:



Si vrem sa realizam o operatie de clusterizare avand valoarea de threshold egala cu 100, imaginea rezultata va fi:



Task 6: Clean Valgrind BONUS (20p)

Intrucat unul din scopurile principale ale acestei teme este alocarea dinamica a memoriei, pentru a rezolva acest task trebuie sa nu aveti nicio eroare sau leak de memorie la rularea utilitarului  valgrind pe aceasta.

Utilitarul se va rula folosind urmatoarea comanda:

```
valgrind --tool=memcheck --leak-check=full --
error-exitcode=1 ./bmp
```

Atentie! Numele executabilului rezultat in urma comenzii



```
make build
```

trebuie sa fie neaparat **bmp**!



Atentie! Pentru a putea primi punctaj pe acest task trebuie sa aveti punctaj maxim pe restul cerintelor!

Format input

Fisierul de intrare, in care se afla informatiile despre fiecare task, se numeste **input.txt**. Acesta are urmatoarea structura:

- Pe prima linie se va gasi numele **fisierului binar** ce reprezinta imaginea in format **bmp** care va fi prelucrata ulterior si are formatul descris la inceput (aceasta are cu siguranta extensia **.bmp**, pentru a va putea fi mai usoara creerea numelor imaginilor de output)
- Pe a doua linie se va gasi numele **fisierului text** ce reprezinta filtrul ce trebuie aplicat la task-ul 3 (semnificatia acestuia este gasita la descrierea task-ului aferent)
- Pe a treia linie se va gasi numele **fisierului text** ce reprezinta filtrul de pooling ce trebuie aplicat la task-ul 4 (semnificatia acestuia este gasita la descrierea task-ului aferent)
- Pe a patra linie se va gasi numele **fisierului text** ce contine valoarea de threshold necesara aplicarii algoritmului de clustering de la task-ul 5

Format filtru:

```
N
VAL_11 VAL_12 ... VAL_1N
VAL_21 VAL_22 ... VAL_2N
.....
VAL_N1 VAL_N2 ... VAL_NN
```


- **N**: dimensiunea filtrului
- **VAL_IJ**: valoarea filtrului de tip **double** de pe linia i si coloana j

Format filtru pooling:

```
m/M N
```

- primul caracter are valoarea **m** daca filtrul este de **minim**, iar **M** daca este de **maxim**
- **N**: dimensiunea filtrului

Exemplu

 input.txt

```
test1.bmp
./input/filters/filter1.txt
./input/pooling/pooling1.txt
./input/clustering/cluster1.txt
```



Exemple de imagini in format bmp si de input-uri pentru task-uri gasiti in directorul `input` din checker.



Cand scrieti imagini in format BMP, intre ultimul byte din header si byte-ul la care incepe matricea de pixeli (adica pana la byte-ul de offset) trebuie sa scrieti peste tot byte-ul 0.

Format output

Dupa cum a fost descris deja, fiecare task are ca scop producerea unei noi imagini, dupa aplicarea unei operatii de procesare pe imaginea initiala. Prin urmare, programul vostru va trebui sa citeasca imaginea de input si sa produca **5** imagini noi, fiecare avand numele specificat in cadrul task-ului corespunzator.

Bineinteles, puteti sa sariti peste implementarea unui task sau sa implementati selectiv un anumit numar de taskuri, singura restrictie pe care o aveti este sa nu primiti erori de tipul Segmentation fault, deoarece acestea vor invalida testarea temei.

Resurse si checker-ul local

Resursele pentru tema se pot descarca de [aici](#). Sunt prezente:

- **bmp_header.h**: headerul care contine declaratiile struct-urilor pe care le veti folosi in citirea unui fisier BMP;
- **checker.zip**: arhiva zip ce contine checker-ul cu care puteti sa va testati implementarea local.

Trimitere tema

Tema va fi trimisa folosind [v2.vmchecker](#), cursul

Programarea Calculatoarelor, tema **Image Processing**.

Punctajul:

- 110p - teste (20p sunt bonus)
 - 10 teste
 - 9p pe test
 - 20p pentru rularea fara erori a utilitarului valgrind pe imaginea cea mai mare (doar daca aveti deja 90p)
- 10p - coding style & README (checker-ul o sa va acorde aceste puncte doar pentru prezenta fisierului README, la corectare acestea vor fi validate)

Restrictii si precizari

- Nu folositi variabile globale.
- Fiti consistenti in ceea ce priveste **Coding Style-ul**.
- Toate matricile si vectorii folositi se vor aloca **dinamic**, pe **heap**;
- **Task-ul 2** este singurul in care aveti de modificat si headerele imaginii bmp, in rest **nu veti avea de modificat nimic in headerele imaginii**;
- Tema se va trimite pe vmchecker si se va testa local cu ajutorul checker-ului care va fi disponibil **in curand**;
- Pe vmchecker veti uploada o arhiva in format .zip care sa contina:
 1. **Makefile**, cu cel puțin 3 targeturi, **build**, **run** și **clean**; Regula **run** trebuie sa ruleze executabilul care a fost obtinut la regula **build** si care are numele **bmp** (vezi si task-ul bonus);
 2. **sursele** voastre, adică fișierele .c și .h. **Inclusiv headerul bmp_header.h sau sub orice altă denumire îl folosiți**;
 3. **README**, în care trebuie să dați detalii despre implementare, de ce ați ales să rezolvați într-un anumit fel, etc.
- Daca rezolvati doar o parte din task-uri asigurati-va ca pe celelalte nu primiti erori la rulare (precum SEGFAULT) sau time limit exceeded, altfel tot testul va fi punctat cu 0. De exemplu, daca task-urile 1 si 2 sunt OK dar task-ul 3 primeste SEGFAULT sau dureaza prea mult atunci tot testul se va nota cu 0.



Temele care nu folosesc alocare dinamica vor avea o depunctare de pana la 50 de puncte din punctajul temei, in functie de gravitate! De asemenea, nu va fi acordat punctajul pe bonus in acest caz!



Nu descarcati imaginile atasate in enunt pentru testare! Acestea nu respecta formatul BMP! Pentru testare folositi **numai** imaginile puse la dispozitie in arhiva de testare. Imaginile de mai sus au doar caracter informativ.

Listă depunctări

Lista nu este exhaustivă. Se pot aplica chiar depunctări mai mari în cazuri excepționale.

- o temă care nu compilează și nu a rulat pe **vmchecker** nu va fi luată în considerare
- o temă care nu rezolvă cerința și trece testele prin alte mijloace nu va fi luată în considerare
- [-50.0]: o tema care nu folosește deloc alocarea dinamică (-70.0 dacă aceasta ia și punctajul bonus)
- [-1.0]: warning-uri la compilare (este obligatorie folosirea în fișierul **Makefile** a flag-ului de compilare **-Wall** pentru regula **build**)
- [-1.0]: numele variabilelor nu sunt sugestive
- [-1.0]: linii mai lungi de 80 de caractere in cod / in fișierul README
- [-1.0]: cod nemodular, funcții prea lungi (inclusiv main)
- [-2.5]: variabile globale
- [-5.0]: abordare ineficientă
 - în cadrul cursului de programare nu avem ca obiectiv rezolvarea în cel mai eficient mod posibil a programelor; totuși, ne dorim ca abordarea să nu fie una ineficientă, de genul să nu folosiți instrucțiuni repetitive acolo unde clar nu era cazul, etc.

[programare/teme_2019/tema3_2019_cbd.txt](#) · Last modified: 2019/12/20 10:27 by laurentiu.stefan97

[Old revisions](#)

[Media Manager](#)

[Back to top](#)