

# **Wprowadzenie do przetwarzania obrazów - sprawozdanie z projektu**

**Juliusz Stańczyk**



# **Spis treści**

<b>1 Wstęp</b>	<b>6</b>
<b>2 Uruchamianie aplikacji i dane techniczne</b>	<b>6</b>
<b>3 Opis aplikacji</b>	<b>6</b>
3.1 Błędy/niedogodności w obsłudze . . . . .	6
<b>4 Funkcja normalizacyjna</b>	<b>9</b>
<b>5 Operacje ujednolicania obrazów</b>	<b>9</b>
5.1 Ujednolicenie obrazów szarych geometrycznie . . . . .	9
5.1.1 Opis działania . . . . .	9
5.1.2 Ograniczania, uwagi . . . . .	10
5.1.3 Możliwe problemy . . . . .	10
5.1.4 Przykładowe wykonanie . . . . .	10
5.2 Ujednolicenie obrazów szarych rozdzielczościowe . . . . .	13
5.2.1 Opis działania . . . . .	13
5.2.2 Ograniczania, uwagi . . . . .	13
5.2.3 Możliwe problemy . . . . .	13
5.2.4 Przykładowe wykonanie . . . . .	14
5.3 Ujednolicenie obrazów barwowych geometrycznie . . . . .	17
5.3.1 Opis działania . . . . .	17
5.3.2 Ograniczania, uwagi . . . . .	18
5.3.3 Możliwe problemy . . . . .	18
5.3.4 Przykładowe wykonanie . . . . .	18
5.4 Ujednolicenie obrazów barwowych rozdzielczościowe . . . . .	21
5.4.1 Opis działania . . . . .	21
5.4.2 Ograniczania, uwagi . . . . .	21
5.4.3 Możliwe problemy . . . . .	21
5.4.4 Przykładowe wykonanie . . . . .	22
<b>6 Operacje sumowania arytmetycznego obrazów szarych</b>	<b>25</b>
6.1 Sumowanie stałej z obrazem oraz dwóch obrazów . . . . .	25
6.1.1 Opis działania . . . . .	25
6.1.2 Ograniczania, uwagi . . . . .	26
6.1.3 Możliwe problemy . . . . .	26
6.1.4 Przykładowe wykonanie . . . . .	26
6.2 Mnożenie obrazu przez zadaną liczbę oraz przez inny obraz . . . . .	31
6.2.1 Opis działania . . . . .	31
6.2.2 Ograniczania, uwagi . . . . .	31
6.2.3 Możliwe problemy . . . . .	31
6.2.4 Przykładowe wykonanie . . . . .	31
6.3 Mieszanie obrazów z określonym współczynnikiem . . . . .	34
6.3.1 Opis działania . . . . .	34

6.3.2	Ograniczania, uwagi . . . . .	34
6.3.3	Możliwe problemy . . . . .	34
6.3.4	Przykładowe wykonanie . . . . .	34
6.4	Potęgowanie obrazu z zadaną potęgą . . . . .	37
6.4.1	Opis działania . . . . .	37
6.4.2	Ograniczania, uwagi . . . . .	37
6.4.3	Możliwe problemy . . . . .	37
6.4.4	Przykładowe wykonanie . . . . .	37
6.5	Dzielenie obrazu przez liczbę oraz przez inny obraz . . . . .	40
6.5.1	Opis działania . . . . .	40
6.5.2	Ograniczania, uwagi . . . . .	40
6.5.3	Możliwe problemy . . . . .	40
6.5.4	Przykładowe wykonanie . . . . .	40
6.6	Pierwiastkowanie obrazu . . . . .	43
6.6.1	Opis działania . . . . .	43
6.6.2	Ograniczania, uwagi . . . . .	43
6.6.3	Możliwe problemy . . . . .	43
6.6.4	Przykładowe wykonanie . . . . .	43
6.7	Logarytmowanie obrazu . . . . .	46
6.7.1	Opis działania . . . . .	46
6.7.2	Ograniczania, uwagi . . . . .	46
6.7.3	Możliwe problemy . . . . .	46
6.7.4	Przykładowe wykonanie . . . . .	46
<b>7</b>	<b>Operacje sumowania arytmetycznego obrazów barwowych</b>	<b>49</b>
7.1	Sumowanie stałej z obrazem oraz dwóch obrazów . . . . .	49
7.1.1	Opis działania . . . . .	49
7.1.2	Ograniczania, uwagi . . . . .	50
7.1.3	Możliwe problemy . . . . .	50
7.1.4	Przykładowe wykonanie . . . . .	50
7.2	Mnożenie obrazu przez zadaną liczbę oraz przez inny obraz . . . . .	55
7.2.1	Opis działania . . . . .	55
7.2.2	Ograniczania, uwagi . . . . .	55
7.2.3	Możliwe problemy . . . . .	55
7.2.4	Przykładowe wykonanie . . . . .	55
7.3	Mieszanie obrazów z określonym współczynnikiem . . . . .	58
7.3.1	Opis działania . . . . .	58
7.3.2	Ograniczania, uwagi . . . . .	58
7.3.3	Możliwe problemy . . . . .	58
7.3.4	Przykładowe wykonanie . . . . .	58
7.4	Potęgowanie obrazu z zadaną potęgą . . . . .	61
7.4.1	Opis działania . . . . .	61
7.4.2	Ograniczania, uwagi . . . . .	61
7.4.3	Możliwe problemy . . . . .	61
7.4.4	Przykładowe wykonanie . . . . .	61
7.5	Dzielenie obrazu przez liczbę oraz przez inny obraz . . . . .	64

7.5.1	Opis działania . . . . .	64
7.5.2	Ograniczania, uwagi . . . . .	64
7.5.3	Możliwe problemy . . . . .	64
7.5.4	Przykładowe wykonanie . . . . .	64
7.6	Pierwiastkowanie obrazu . . . . .	67
7.6.1	Opis działania . . . . .	67
7.6.2	Ograniczania, uwagi . . . . .	67
7.6.3	Możliwe problemy . . . . .	67
7.6.4	Przykładowe wykonanie . . . . .	67
7.7	Logarytmowanie obrazu . . . . .	70
7.7.1	Opis działania . . . . .	70
7.7.2	Ograniczania, uwagi . . . . .	70
7.7.3	Możliwe problemy . . . . .	70
7.7.4	Przykładowe wykonanie . . . . .	70
<b>8</b>	<b>Operacje geometryczne na obrazie</b>	<b>73</b>
8.1	Przemieszczanie obrazu o zadany wektor . . . . .	73
8.1.1	Opis działania . . . . .	73
8.1.2	Ograniczania, uwagi . . . . .	73
8.1.3	Możliwe problemy . . . . .	73
8.1.4	Przykładowe wykonanie . . . . .	73
8.2	Jednorodne i niejednorodne skalowanie obrazu . . . . .	75
8.2.1	Opis działania . . . . .	75
8.2.2	Ograniczania, uwagi . . . . .	75
8.2.3	Możliwe problemy . . . . .	75
8.2.4	Przykładowe wykonanie . . . . .	75
8.3	Obracanie obrazu o dowolny kąt . . . . .	77
8.3.1	Opis działania . . . . .	77
8.3.2	Ograniczania, uwagi . . . . .	77
8.3.3	Możliwe problemy . . . . .	78
8.3.4	Przykładowe wykonanie . . . . .	78
8.4	Symetrie względem osi układu i zadanej prostej . . . . .	80
8.4.1	Opis działania . . . . .	80
8.4.2	Ograniczania, uwagi . . . . .	81
8.4.3	Możliwe problemy . . . . .	81
8.4.4	Przykładowe wykonanie . . . . .	81
8.5	Wycinanie fragmentów obrazu . . . . .	84
8.5.1	Opis działania . . . . .	84
8.5.2	Ograniczania, uwagi . . . . .	84
8.5.3	Możliwe problemy . . . . .	84
8.5.4	Przykładowe wykonanie . . . . .	84
8.6	Wycinanie fragmentów obrazu . . . . .	86
8.6.1	Opis działania . . . . .	86
8.6.2	Ograniczania, uwagi . . . . .	86
8.6.3	Możliwe problemy . . . . .	86
8.6.4	Przykładowe wykonanie . . . . .	86

<b>9 Operacje na histogramie obrazu szarego</b>	<b>88</b>
9.1 Obliczanie histogramu . . . . .	88
9.1.1 Opis działania . . . . .	88
9.1.2 Ograniczania, uwagi . . . . .	88
9.1.3 Możliwe problemy . . . . .	88
9.1.4 Przykładowe wykonanie . . . . .	88
9.2 Przemieszczanie histogramu . . . . .	91
9.2.1 Opis działania . . . . .	91
9.2.2 Ograniczania, uwagi . . . . .	91
9.2.3 Możliwe problemy . . . . .	91
9.2.4 Przykładowe wykonanie . . . . .	91
9.3 Rozciąganie histogramu . . . . .	94
9.3.1 Opis działania . . . . .	94
9.3.2 Ograniczania, uwagi . . . . .	94
9.3.3 Możliwe problemy . . . . .	94
9.3.4 Przykładowe wykonanie . . . . .	94
9.4 Progowanie lokalne . . . . .	97
9.4.1 Opis działania . . . . .	97
9.4.2 Ograniczania, uwagi . . . . .	97
9.4.3 Możliwe problemy . . . . .	97
9.4.4 Przykładowe wykonanie . . . . .	97
9.5 Progowanie globalne . . . . .	99
9.5.1 Opis działania . . . . .	99
9.5.2 Ograniczania, uwagi . . . . .	99
9.5.3 Możliwe problemy . . . . .	99
9.5.4 Przykładowe wykonanie . . . . .	99
<b>10 Operacje na histogramie obrazu barwowego</b>	<b>101</b>
10.1 Obliczanie histogramu . . . . .	101
10.1.1 Opis działania . . . . .	101
10.1.2 Ograniczania, uwagi . . . . .	101
10.1.3 Możliwe problemy . . . . .	101
10.1.4 Przykładowe wykonanie . . . . .	101
10.2 Przemieszczanie histogramu . . . . .	104
10.2.1 Opis działania . . . . .	104
10.2.2 Ograniczania, uwagi . . . . .	104
10.2.3 Możliwe problemy . . . . .	104
10.2.4 Przykładowe wykonanie . . . . .	104
10.3 Rozciąganie histogramu . . . . .	107
10.3.1 Opis działania . . . . .	107
10.3.2 Ograniczania, uwagi . . . . .	107
10.3.3 Możliwe problemy . . . . .	107
10.3.4 Przykładowe wykonanie . . . . .	107
10.4 Progowanie globalne wieloprogowe . . . . .	110
10.4.1 Opis działania . . . . .	110
10.4.2 Ograniczania, uwagi . . . . .	110

10.4.3	Możliwe problemy . . . . .	110
10.4.4	Przykładowe wykonanie . . . . .	110
10.5	Progowanie lokalne wieloprogowe . . . . .	112
10.5.1	Opis działania . . . . .	112
10.5.2	Ograniczania, uwagi . . . . .	112
10.5.3	Możliwe problemy . . . . .	112
10.5.4	Przykładowe wykonanie . . . . .	112
10.6	Progowanie lokalne jednoprogowe . . . . .	114
10.6.1	Opis działania . . . . .	114
10.6.2	Ograniczania, uwagi . . . . .	114
10.6.3	Możliwe problemy . . . . .	114
10.6.4	Przykładowe wykonanie . . . . .	114
10.7	Progowanie globalne jednoprogowe . . . . .	116
10.7.1	Opis działania . . . . .	116
10.7.2	Ograniczania, uwagi . . . . .	116
10.7.3	Możliwe problemy . . . . .	116
10.7.4	Przykładowe wykonanie . . . . .	116

## **1 Wstęp**

Opis programu na zaliczenie przedmiotu Przetwarzanie obrazów. Składa się z opisu uruchomienia, obsługi i działania aplikacji oraz opisu zastosowanych algorytmów. Dokument zawiera również opis wszystkich wiadomych mi nieścisłości z poleceniami oraz błędów algorytmów/aplikacji.

## **2 Uruchamianie aplikacji i dane techniczne**

Aplikacja została napisana w języku Python 3.7 i w tej wersji została testowana.

Środowiskiem działania aplikacji jest PyCharm w wersji 2019.2.

Aplikacje można uruchomić poprzez uruchomienie PyCharm i wybranie jej folder (File - Open - "wybór aplikacji") i uruchomienie. Powinna być gotowa do użycia po naciśnięciu przycisku "Run" w prawym górnym rogu.

## **3 Opis aplikacji**

Aplikacja posiada trzy strony z przyciskami odpowiedzialnymi za uruchomienie konkretnej funkcji - zadania oraz z przyciskami do przechodzenia na kolejne/poprzednie strony.

Każda podstrona konkretnego podpunktu posiada różne przyciski lub okna wejściowe dla danych potrzebnych do wykonania zadania. Przed rozpoczęciem wykonywania operacji należy wybrać obrazy oraz wypełnić przygotowane pola (jeśli są dostępne). Dopiero potem należy kliknąć przycisk "wykonaj operacje".

Przycisk powinien być podświetlony przez czas trwania operacji max kilka-naście sekund). Jeśli przycisk zgasł, oznacza to, że coś poszło nie tak. W tym przypadku najlepiej wrócić do menu i ponowne wejść w daną podstronę. Ta operacja zresetuje ją.

Aplikacja oprócz wyświetlania obrazów zapisuje je w pliku w którym się znajduje. W przypadku tworzenia histogramów, obraz utworzony przez mnie ręcznie zostaje zapisany do pliku. Nie jest on wyświetlany w aplikacji, ponieważ nie udało mi się tego zrobić. Zostaje natomiast wyświetlony histogram za pomocą funkcji tworzącej wykres z biblioteki numpy, z przygotowanych przez mnie danych. Wyświetlony histogram również można zapisać, w ramach funkcjonalności biblioteki.

Do wybrania z pliku będą możliwe tylko pliki z końcówką ".png".

W polach na wprowadzanie danych, liczby dziesiętne powinny być wprowadzane z kropką.

### **3.1 Błędy/niedogodności w obsłudze**

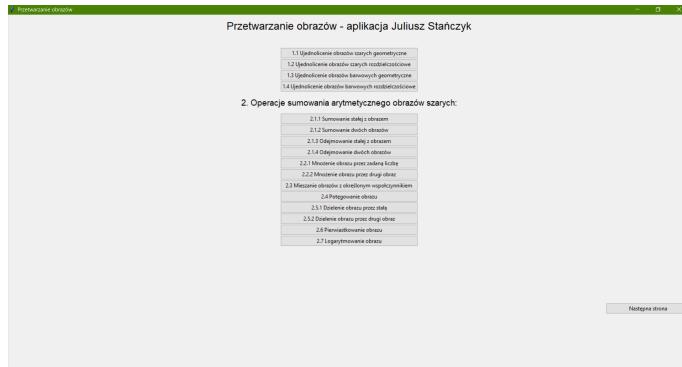
1. Po kliknięciu przycisku "Wybierz obraz", wybraniu obrazu i po ponownym kliknięciu tego samego przycisku i wybraniu obrazu wystąpi mały błąd wizual-

ny. Nowo wybrany obraz będzie lekko przesunięty.

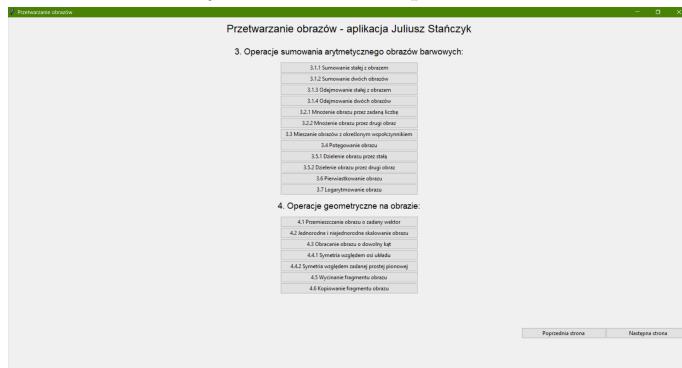
2. Bez wybrania obrazów lub wypełniania pól liczbowych i kliknięciu przycisku "Wykonaj operacje" może wystąpić błąd. W przypadku wcześniejszego uruchomienia tej funkcji do operacji zostaną wzięte stare obrazy. W przeciwnym przypadku program nie wykona żadnej operacji.

3. Po każdej wykonanej operacji (z sukcesem lub bez) trzeba wrócić do menu i następnie ponownie włączyć daną funkcję. Tzn. niemożna jednej funkcji wykonać dwa razy pod rząd (bez resetu), ponieważ wyświetlane obrazy się nie usuwają.

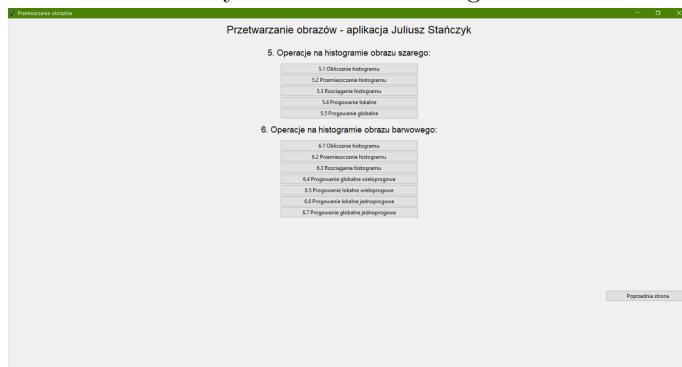
4. Większe obrazy wynikowe mogą się nie mieścić w ekranie aplikacji, zależy od rozmiaru monitora.



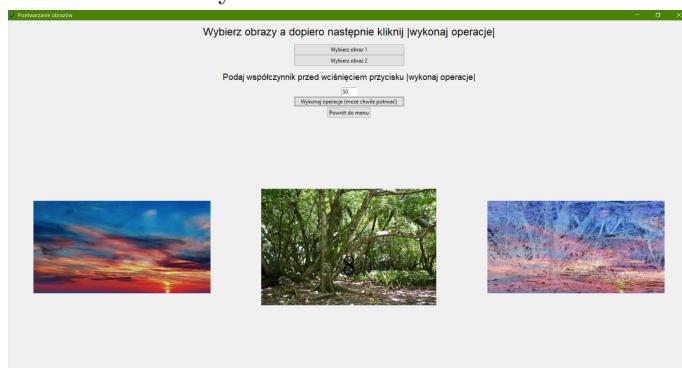
Rysunek 1: Strona pierwsza



Rysunek 2: Strona druga



Rysunek 3: Strona trzecia



Rysunek 4: Przykładowa podstrona z wykonaniem

## 4 Funkcja normalizacyjna

Algorytm tworzy trzy tablice do których zapisuje osobny kanał RGB. Następnie wykrywa min i max wartość dla wszystkich kanałów. Następnie tworzy trzy tablice pomocnicze, dla każdego kanału po jedną. Dla każdego piksla wykonuje:  $(wartosc piksla - min) / (max - min) * 255$ . Następnie kanały są łączone i obraz jest zwracany.

## 5 Operacje ujednoliciania obrazów

### 5.1 Ujednolicenie obrazów szarych geometrycznie

#### 5.1.1 Opis działania

Algorytm rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamknie. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program ujednolica wysokość i szerokość obrazów. Algorytm składa się z czterech instrukcji warunkowych, wykonanych zależnie od rozmiarów obrazów. Height i width oznaczają wysokość i szerokość obrazu pierwszego lub drugiego, zależnie od numeru.

1.  $height1 \leq height2 \text{ and } width1 \leq width2$
2.  $height1 \geq height2 \text{ and } width1 \geq width2$
3.  $height1 \geq height2 \text{ and } width1 \leq width2$
4.  $height1 \leq height2 \text{ and } width1 \geq width2$

Pierwszy warunek oznacza, że obraz drugi jest większy od pierwszego w wysokości i szerokości. Algorytm tworzy tablice wynikową o rozmiarach obrazu drugiego, do której zostanie dodany obraz pierwszy. Zrobi to przehodząc po kolejnych pikslach obrazu, dla rozmiarów obrazu większego. Dla każdego piksla obrazu pierwszego, mieszczącego się w swoich rozmiarach, przypisze do tablicy wynikowej piksel z obrazu pierwszego. Dla reszty piksli przypisze wartość 1 w celu uniknięcia ewentualnych problemów przy dzieleniu obrazu. Zapisze obraz do pliku *img1.1(gotowy).png*.

Drugi warunek jest analogiczny do pierwszego. W tym przypadku obraz pierwszy jest większy. Algorytm działa jak wyżej, ale obraz pierwszy zostaje zastąpiony drugim i odwrotnie. Zapisze obraz do pliku *img1.1(gotowy).png*.

Trzeci warunek rozwiązuje sytuację, gdy obraz pierwszy ma większą wysokość, ale mniejszą szerokość. Algorytm wyszukuje maksymalną wysokość i szerokość z obu obrazów i tworzy dla nich dwie tablice wynikowe. Następne operacje

wykonuje dla każdego piksla, w pętlach o wielkości maksymalnych, obliczonych przed chwilą. Dla obrazu drugiego, mieszczącego się w swoim rozmiarze, przypisze do drugiej tablicy wynikowej piksele obrazu drugiego. W przypadku nie-mieszczenia się w rozmiarze przypisze wartość 1 w celu umiknięcia ewentualnych problemów przy dzieleniu obrazu. Analogicznie zrobi dla obrazu pierwszego, mieszczącego się i niemieszczącego się w swoim rozmiarze. Zapisze obraz do pliku *img1.1 – 1(gotowy).png* oraz *img1.1 – 2(gotowy).png*.

Czwarty warunek działa analogicznie do trzeciego. W tym przypadku wysokość pierwszego obrazu jest mniejsza a szerokość większa. Algorytm działa jak wyżej. Zapisze obraz do pliku '*img1.1 – 1(gotowy).png*' oraz *img1.1 – 2(gotowy).png*.

### **5.1.2 Ograniczania, uwagi**

Obraz mniejszy lub oba obrazy będą miały czarne "ramki", ale takie jest założenie zadania.

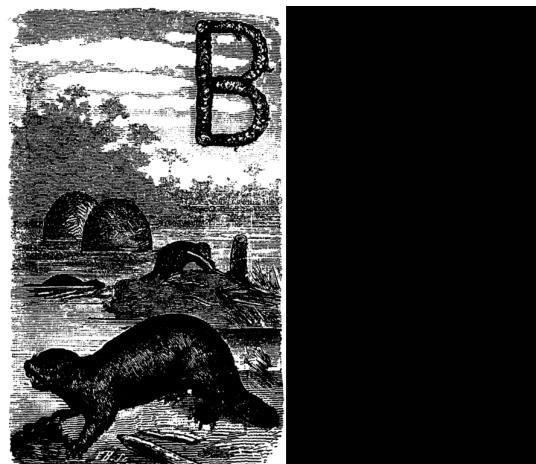
### **5.1.3 Możliwe problemy**

-

### **5.1.4 Przykładowe wykonanie**



Rysunek 5: Przed (wyższy)



Rysunek 6: Po



Rysunek 7: Przed (szerszy)



Rysunek 8: Po



Rysunek 9: Przed



Rysunek 10: Przed (większy)



Rysunek 11: Po

## 5.2 Ujednolicenie obrazów szarych rozdzielczościowe

### 5.2.1 Opis działania

Algorytm rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm składa się z czterech instrukcji warunkowych, wykonanych zależnie od rozmiarów obrazów. Height i width oznaczają wysokość i szerokość obrazu pierwszego lub drugiego, zależnie od numeru.

1.  $height1 \leq height2 \text{ and } width1 \leq width2$
2.  $height1 \geq height2 \text{ and } width1 \geq width2$
3.  $height1 \geq height2 \text{ and } width1 \leq width2$
4.  $height1 \leq height2 \text{ and } width1 \geq width2$

Pierwszy warunek oznacza, że obraz drugi jest większy od pierwszego w wysokości i szerokości. Algorytm pobiera współczynnik wysokości i szerokości większego obrazu do mniejszego i na ich podstawie wykonuje funkcję skalowania z zadania 4.2 (dokładny opis jest w późniejszym etapie dokumentacji). Algorytm zapisuje obraz jako *img1.2(gotowy).png*

Drugi warunek jest analogiczny do pierwszego. W tym przypadku obraz pierwszy jest większy. Obraz wynikowy zapisze obraz do pliku *img1.2(gotowy).png*.

Trzeci warunek rozwiązuje sytuacje, gdy obraz pierwszy ma większą wysokość, ale mniejszą szerokość. Algorytm znowu policzy współczynnik wysokości i szerokości większego obrazu do mniejszego. W tym przypadku oba obrazy zostaną zmienione i zapisane pod nazwami *img1.2 – 1(gotowy).png* oraz *img1.2 – 2(gotowy).png*

Czwarty warunek działa analogicznie do trzeciego. W tym przypadku wysokość pierwszego obrazu jest mniejsza a szerokość większa. Zapisze obrazy do pliku '*img1.2 – 1(gotowy).png*' oraz *img1.2 – 2(gotowy).png*.

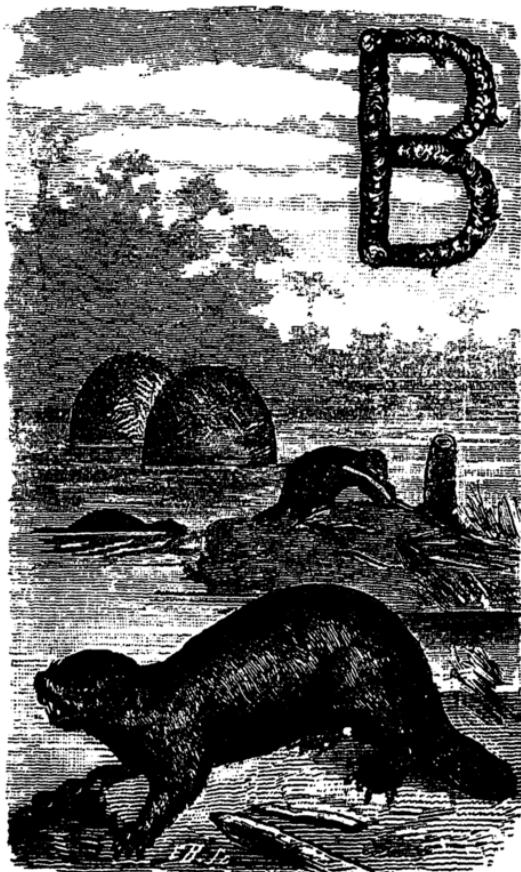
### 5.2.2 Ograniczania, uwagi

Nie jestem pewny czy dobrze zinterpretowałem polecenie.

### 5.2.3 Możliwe problemy

-

#### **5.2.4 Przykładowe wykonanie**



Rysunek 12: Przed (wyższy)

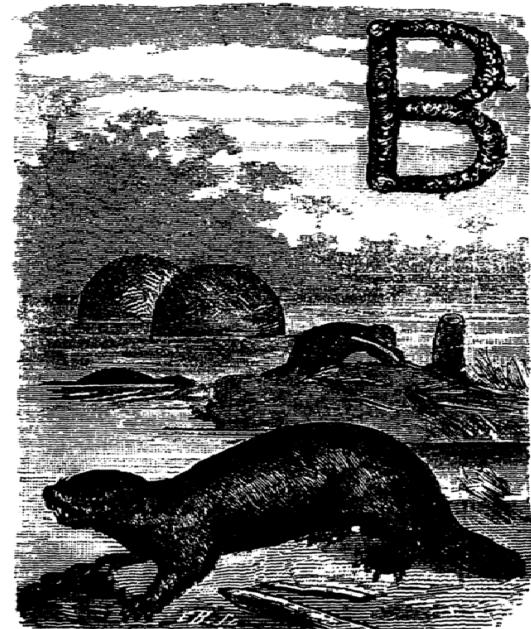


Rysunek 13: Po



15

Rysunek 14: Przed (szerszy)



Rysunek 15: Po



Rysunek 16: Przed



Rysunek 17: Przed (większy)



Rysunek 18: Po

## 5.3 Ujednolicenie obrazów barwowych geometrycznie

### 5.3.1 Opis działania

Algorytm rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program ujednolica wysokość i szerokość obrazów. Algorytm składa się z czterech instrukcji warunkowych, wykonanych zależnie od rozmiarów obrazów. Height i width oznaczają wysokość i szerokość obrazu pierwszego lub drugiego, zależnie od numeru.

1.  $height1 \leq height2 \text{ and } width1 \leq width2$
2.  $height1 \geq height2 \text{ and } width1 \geq width2$
3.  $height1 \geq height2 \text{ and } width1 \leq width2$
4.  $height1 \leq height2 \text{ and } width1 \geq width2$

Pierwszy warunek oznacza, że obraz drugi jest większy od pierwszego w wysokości i szerokości. Algorytm tworzy tablice wynikowa o rozmiarach obrazu drugiego, do której zostanie dodany obraz pierwszy. Zrobi to przechodząc po kolejnych pikslach obrazu, dla rozmiarów obrazu większego. Dla każdego piksla obrazu pierwszego, mieszczącego się w swoich rozmiarach, przypisze do tablicy wynikowej piksel z obrazu pierwszego. Dla reszty piksli przypisze wartość 1 w celu uniknięcia ewentualnych problemów przy dzieleniu obrazu. Zapisze obraz do pliku *img1.3(gotowy).png*.

Drugi warunek jest analogiczny do pierwszego. W tym przypadku obraz pierwszy jest większy. Algorytm działa jak wyżej, ale obraz pierwszy zostaje zastąpiony drugim i odwrotnie. Zapisze obraz do pliku *img1.3(gotowy).png*.

Trzeci warunek rozwiązuje sytuacje, gdy obraz pierwszy ma większą wysokość, ale mniejszą szerokość. Algorytm wyszukuje maksymalna wysokość i szerokość z obu obrazów i tworzy dla nich dwie tablice wynikowe. Następne operacje wykonuje dla każdego piksla, w pętlach o wielkości maksymalnych, obliczonych przed chwilą. Dla obrazu drugiego, mieszczącego się w swoim rozmiarze, przypisze do drugiej tablicy wynikowej piksle obrazu drugiego. W przypadku nieszczycenia się w rozmiarze przypisze wartość 1 w celu uniknięcia ewentualnych problemów przy dzieleniu obrazu. Analogicznie zrobi dla obrazu pierwszego, mieszczącego się i niemieszczącego się w swoim rozmiarze. Zapisze obraz do pliku *img1.3 – 1(gotowy).png* oraz *img1.3 – 2(gotowy).png*.

Czwarty warunek działa analogicznie do trzeciego. W tym przypadku wysokość pierwszego obrazu jest mniejsza a szerokość większa. Algorytm dzia-

ła jak wyżej. Zapisze obraz do pliku '*img1.3 – 1(gotowy).png*' oraz *img1.3 – 2(gotowy).png*.

### **5.3.2 Ograniczania, uwagi**

Obraz mniejszy lub oba obrazy będą miały czarne ”ramki”, ale takie jest założenie zadania.

### **5.3.3 Możliwe problemy**

-

### **5.3.4 Przykładowe wykonanie**



Rysunek 19: Przed (wyższy)



Rysunek 20: Po



Rysunek 21: Przed (szerszy)



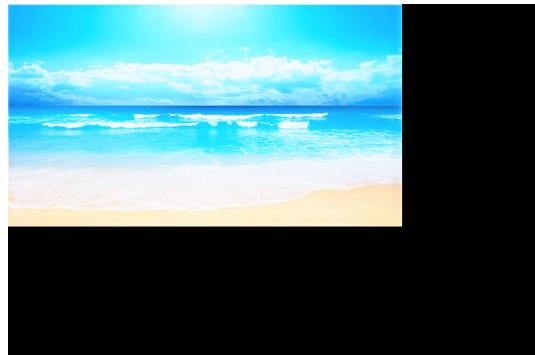
Rysunek 22: Po



Rysunek 23: Przed



Rysunek 24: Przed (większy)



Rysunek 25: Po

## 5.4 Ujednolicenie obrazów barwowych rozdzielczościowe

### 5.4.1 Opis działania

Algorytm rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm składa się z czterech instrukcji warunkowych, wykonanych zależnie od rozmiarów obrazów. Height i width oznaczają wysokość i szerokość obrazu pierwszego lub drugiego, zależnie od numeru.

1.  $height1 \leq height2 \text{ and } width1 \leq width2$
2.  $height1 \geq height2 \text{ and } width1 \geq width2$
3.  $height1 \geq height2 \text{ and } width1 \leq width2$
4.  $height1 \leq height2 \text{ and } width1 \geq width2$

Pierwszy warunek oznacza, że obraz drugi jest większy od pierwszego w wysokości i szerokości. Algorytm pobiera współczynnik wysokości i szerokości większego obrazu do mniejszego i na ich podstawie wykonuje funkcję skalowania z zadania 4.2 (dokładny opis jest w późniejszym etapie dokumentacji). Algorytm zapisuje obraz jako *img1.4(gotowy).png*

Drugi warunek jest analogiczny do pierwszego. W tym przypadku obraz pierwszy jest większy. Obraz wynikowy zapisze obraz do pliku *img1.4(gotowy).png*.

Trzeci warunek rozwiązuje sytuacje, gdy obraz pierwszy ma większą wysokość, ale mniejszą szerokość. Algorytm znowu policzy współczynnik wysokości i szerokości większego obrazu do mniejszego. W tym przypadku oba obrazy zostaną zmienione i zapisane pod nazwami *img1.4 – 1(gotowy).png* oraz *img1.4 – 2(gotowy).png*

Czwarty warunek działa analogicznie do trzeciego. W tym przypadku wysokość pierwszego obrazu jest mniejsza a szerokość większa. Zapisze obrazy do pliku '*img1.4 – 1(gotowy).png*' oraz *img1.4 – 2(gotowy).png*.

### 5.4.2 Ograniczania, uwagi

Nie jestem pewny czy dobrze zinterpretowałem polecenie.

### 5.4.3 Możliwe problemy

-

#### **5.4.4 Przykładowe wykonanie**



Rysunek 26: Przed (wyższy)



Rysunek 27: Po



Rysunek 28: Przed (szerszy)



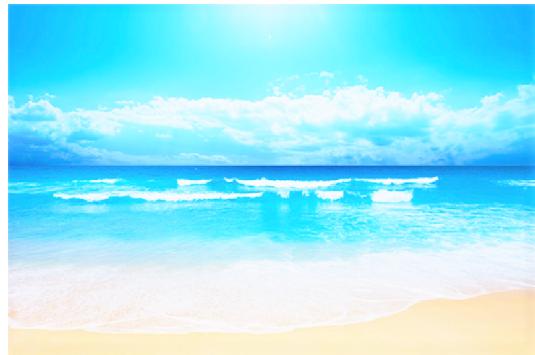
Rysunek 29: Po



Rysunek 30: Przed



Rysunek 31: Przed (większy)



Rysunek 32: Po

## 6 Operacje sumowania arytmetycznego obrazów szarych

W przypadku wgrania dwóch obrazów o różnych rozmiarach zostają one wyrównane do rozmiaru mniejszego obrazu, za pomocą gotowej funkcji bibliotecznej. Zdecydowałem się na takie rozwiązanie ze względu na zminimalizowanie błędów i wcześniejsze pokazanie, że umiem stworzyć taką funkcję (w zadaniu 1.2 i 1.4).

### 6.1 Sumowanie stałej z obrazem oraz dwóch obrazów

#### 6.1.1 Opis działania

Rozwiążanie składa się z czterech osobnych algorytmów.

Pierwszy z nich jest algorytmem sumującym dwa obrazy szare, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego dodaje piksel obrazu pierwszego i drugiego. Jeśli jednak ich suma jest większa niż 255 (maksymalna wartość piksla) wykonuje dodatkowe akcje. W tym przypadku bierze sumę i odejmuje od niej 255 oraz dzielinią przez 255 zaokrąglając w góre (wartość  $x$ ). Następnie zamiast wcześniej przypisanej piksla, na jego miejsce wchodzi piksel o wartości:  $PikselObrazuPierwszego - (PikselObrazuPierwszego * x) + PikselObrazuDrugiego - (PikselObrazuDrugiego * x)$

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.1.1(gotowy).png*.

Drugim algorytmem jest sumowanie obrazu szarego ze stałą. Wszystko odbywa się analogicznie jak wyżej tylko, że dla jednego obrazu. Drugi obraz zastąpiony jest pobraną przez funkcję stałą. Pierwszy kanał obrazu jest zapisywany jako *img2.1.2(gotowy).png*.

Trzecim algorytmem jest odejmowanie dwóch obrazów szarych. Program również analogiczny do pierwszego, z różnicą zmiany znaku na odejmowanie piksla obrazu drugiego od piksla obrazu pierwszego. Pierwszy kanał obrazu jest zapisywany jako *img2.1.3(gotowy).png*.

Czwarty algorytm odejmowania stałej od obrazu szarego analogiczny do drugiego. Różnica polega na odjęciu stałej a nie dodaniu jej. Pierwszy kanał obrazu

jest zapisywany jako *img2.1.4(gotowy).png*.

#### **6.1.2 Ograniczania, uwagi**

-

#### **6.1.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

#### **6.1.4 Przykładowe wykonanie**



Rysunek 33: Przed - dodawanie z stałą



Rysunek 34: Po (bez normalizacji, wsp=50)



Rysunek 35: Po (z normalizacją, wsp=50)



Rysunek 36: Przed - dodawanie dwóch obrazów



Rysunek 37: Przed - dodawanie dwóch obrazów



Rysunek 38: Po (bez normalizacji)



Rysunek 39: Po (z normalizacją)



Rysunek 40: Przed - odejmowanie z stałą



Rysunek 41: Po (bez normalizacji, wsp=30)



Rysunek 42: Po (z normalizacją, wsp=30)



Rysunek 43: Przed - odejmowanie dwóch obrazów



Rysunek 44: Przed - odejmowanie dwóch obrazów



Rysunek 45: Po (bez normalizacji)



Rysunek 46: Po (z normalizacją)

## **6.2 Mnożenie obrazu przez zadaną liczbę oraz przez inny obraz**

### **6.2.1 Opis działania**

Rozwiążanie składa się z dwóch osobnych algorytmów.

Pierwszy z nich jest algorytmem mnożącym dwa obrazy szare, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje dalsze instrukcje. Jeśli piksel wynosi 255 (maksymalną wartość piksla) to jego wartość zostaje przepisana do tablicy wynikowej. Podobnie jeśli piksel wynosi 0 czyli wartość minimalną. Dla reszty piksli zostaje wykonane mnożenie piksli obrazu pierwszego i drugiego oraz podzielenie ich przez 256. Wartość zostanie zaokrąglona w góre.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img2.2.1(gotowy).png*.

Drugim algorytmem jest mnożenie obrazu szarego ze stałą. Wszystko odbywa się analogicznie jak wyżej tylko, że dla jednego obrazu. Drugi obraz zastąpiony jest pobraną przez funkcję stałą. Pierwszy kanał obrazu jest zapisywany jako *img2.2.2(gotowy).png*.

### **6.2.2 Ograniczania, uwagi**

-

### **6.2.3 Możliwe problemy**

-

### **6.2.4 Przykładowe wykonanie**



Rysunek 47: Przed



Rysunek 48: Po (bez normalizacji, wsp=30)



Rysunek 49: Po (z normalizacją, wsp=30)



Rysunek 50: Przed



Rysunek 51: Przed



Rysunek 52: Po (bez normalizacji, wsp=50)



Rysunek 53: Po (z normalizacją, wsp=50)

## **6.3 Mieszanie obrazów z określonym współczynnikiem**

### **6.3.1 Opis działania**

Algorytm miesza dwa obrazy szare z współczynnikiem, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje algorytm:  $wspoczynnik * PikselObrazuPierwszego + (1 - wspoczynnik) * PikselObrazuDrugiego$ . Wartość jest zaokrąglana w góre.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.3(gotowy).png*.

### **6.3.2 Ograniczania, uwagi**

-

### **6.3.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **6.3.4 Przykładowe wykonanie**



Rysunek 54: Przed



Rysunek 55: Przed



Rysunek 56: Po (bez normalizacji, wsp=30)



Rysunek 57: Po (z normalizacją, wsp=30)



Rysunek 58: Przed



Rysunek 59: Przed



Rysunek 60: Po (bez normalizacji)



Rysunek 61: Po (z normalizacją)

## **6.4 Potęgowanie obrazu z zadaną potęgą**

### **6.4.1 Opis działania**

Algorytm potęguje obraz szary podaną potęgą, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest potęgowany do zadanej potęgi ( $\geq 1$ ) a wynik jest zaokrąglany w góre i zapisywany do tablicy wynikowej.

Po tych czynnościami uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.4(gotowy).png*.

### **6.4.2 Ograniczania, uwagi**

-

### **6.4.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą lub stała będzie ujemna. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **6.4.4 Przykładowe wykonanie**



Rysunek 62: Przed

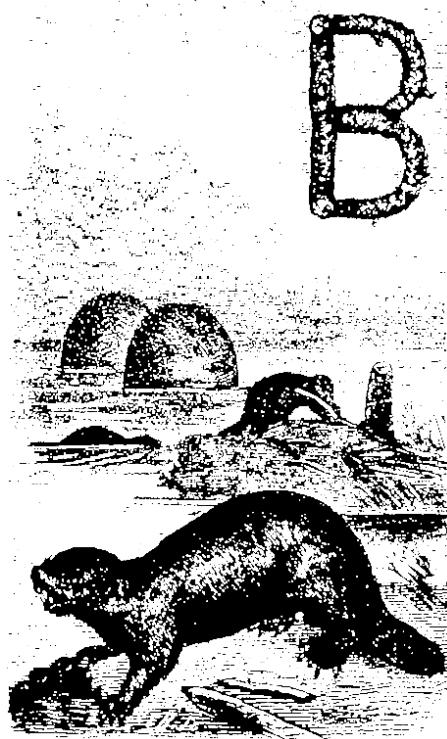
Rysunek 63: Po (bez normalizacji, potęga=2)



Rysunek 64: Po (z normalizacją, potęga=2)



Rysunek 65: Przed



Rysunek 66: Po (bez normalizacji, potega=3)



Rysunek 67: Po (z normalizacją, potega=3)

## **6.5 Dzielenie obrazu przez liczbę oraz przez inny obraz**

### **6.5.1 Opis działania**

Rozwiążanie składa się z dwóch osobnych algorytmów.

Pierwszy algorytm dzieli dwa obrazy szare, rozpoczyna się od wczytania plików. Pliki są otwierane bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje kolejne kroki. Wartości piksla pierwszego i drugiego są dodawane oraz w celu zabezpieczenia przed dzieleniem przez 0 dodawana jest do nich wartość 1. Następnie otrzymana suma jest mnożona przez 255 i dzielona przez opisaną wcześniej  $sum + 1$ . Wynik jest zaokrąglany w górę i zapisywany do tablicy wynikowej.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.5.1(gotowy).png*.

Drugi algorytm działa analogicznie do pierwszego z różnicą, że drugi obraz zastępowany jest stałą. Pierwszy kanał obrazu jest zapisywany jako *img2.5.2(gotowy).png*.

### **6.5.2 Ograniczania, uwagi**

-

### **6.5.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **6.5.4 Przykładowe wykonanie**



Rysunek 68: Przed

Rysunek 69: Po (bez normalizacji, wsp=30)



Rysunek 70: Po (z normalizacją, wsp=30)



Rysunek 71: Przed



Rysunek 72: Przed



Rysunek 73: Po (bez normalizacji)

Rysunek 74: Po (z normalizacją)

## **6.6 Pierwiastkowanie obrazu**

### **6.6.1 Opis działania**

Algorytm pierwiastkuje czyli potęguje obraz szary podaną potęgą, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest potęgowany do zadanej potęgi (z przedziału [0, 1]) a wynik jest zaokrąglany w górę i zapisywany do tablicy wynikowej.

Po tych czynnościami uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.6(gotowy).png*.

### **6.6.2 Ograniczania, uwagi**

Program powinien przyjmować liczby z przedziału [0,1], ale można wpisać też liczby z zakresu [0,4).

### **6.6.3 Możliwe problemy**

Program nie wykona się liczba będzie całkowita lub dziesiętna  $\zeta = 4$  lub  $\mathfrak{j} = 0$ . Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **6.6.4 Przykładowe wykonanie**



Rysunek 75: Przed



Rysunek 76: Po (bez normalizacji, pierw=0.5)



Rysunek 77: Po (z normalizacją, pierw=0.5)



Rysunek 78: Przed



Rysunek 79: Po (bez normalizacji, pierw=0.7)



Rysunek 80: Po (z normalizacją, pierw=0.7)

## **6.7 Logarytmowanie obrazu**

### **6.7.1 Opis działania**

Algorytm logarytmuje obraz szary, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest powiększany o 1 i jest logarytmowany z podstawą logarytmu 10, za pomocą gotowej funkcji matematycznej.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i pierwszy kanał obrazu jest zapisywany jako *img2.7(gotowy).png*.

### **6.7.2 Ograniczania, uwagi**

-

### **6.7.3 Możliwe problemy**

-

### **6.7.4 Przykładowe wykonanie**



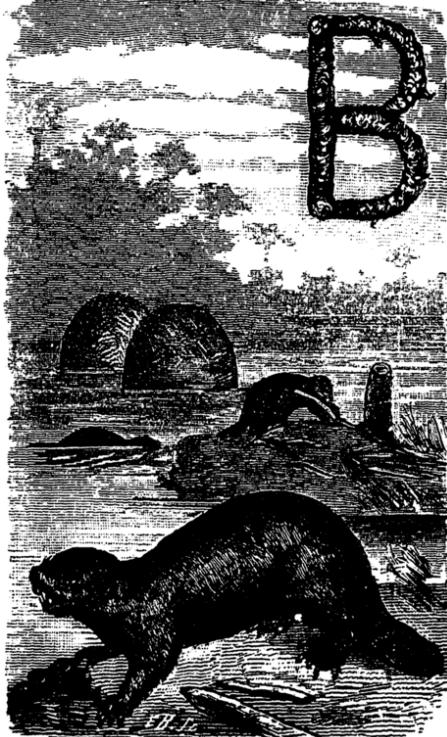
Rysunek 81: Przed



Rysunek 82: Po (bez normalizacji)



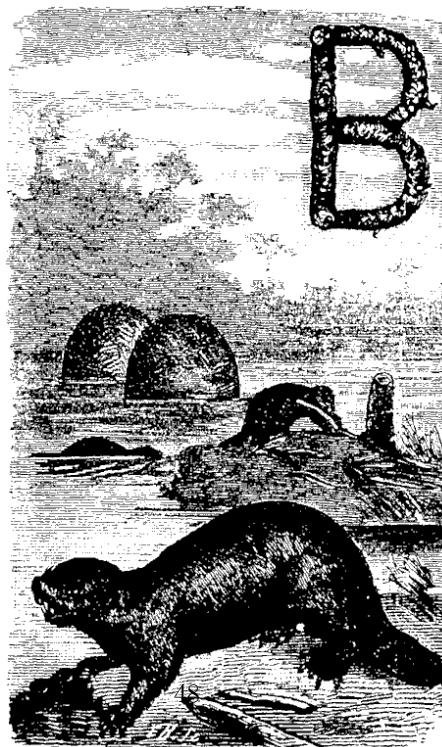
Rysunek 83: Po (z normalizacją)



Rysunek 84: Przed



Rysunek 85: Po (bez normalizacji)



Rysunek 86: Po (z normalizacją)

## 7 Operacje sumowania arytmetycznego obrazów barwowych

W przypadku wgrania dwóch obrazów o różnych rozmiarach zostają one wyrównane do rozmiaru mniejszego obrazu, za pomocą gotowej funkcji bibliotecznej. Zdecydowałem się na takie rozwiązanie ze względu na zminimalizowanie błędów i wcześniejsze pokazanie, że umiem stworzyć taką funkcję (w zadaniu 1.2 i 1.4).

### 7.1 Sumowanie stałej z obrazem oraz dwóch obrazów

#### 7.1.1 Opis działania

Rozwiążanie składa się z czterech osobnych algorytmów.

Pierwszy z nich jest algorytmem sumującym dwa obrazy barwowe, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego dodaje piksel obrazu pierwszego i drugiego. Jeśli jednak ich suma jest większa niż 255 (maksymalna wartość piksla) wykonuje dodatkowe akcje. W tym przypadku bierze sumę i odejmuje od niej 255 oraz dzielinią przez 255 zaokrąglając w góre (wartość  $x$ ). Następnie zamiast wcześniej przypisanej piksla, na jego miejsce wchodzi piksel o wartości:  $PikselObrazuPierwszego - (PikselObrazuPierwszego * x) + PikselObrazuDrugiego - (PikselObrazuDrugiego * x)$

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.1.1(gotowy).png*.

Drugim algorytmem jest sumowanie obrazu szarego ze stałą. Wszystko odbywa się analogicznie jak wyżej tylko, że dla jednego obrazu. Drugi obraz zastąpiony jest pobraną przez funkcję stałą. Obraz jest zapisywany jako *img3.1.2(gotowy).png*.

Trzecim algorytmem jest odejmowanie dwóch obrazów szarych. Program również analogiczny do pierwszego, z różnicą zmiany znaku na odejmowanie piksla obrazu drugiego od piksla obrazu pierwszego. Obraz jest zapisywany jako *img3.1.3(gotowy).png*.

Czwarty algorytm odejmowania stałej od obrazu szarego analogiczny do drugiego. Różnica polega na odjęciu stałej a nie dodaniu jej. Obraz jest zapisywany

jako *img3.1.4(gotowy).png*.

### **7.1.2 Ograniczania, uwagi**

-

### **7.1.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **7.1.4 Przykładowe wykonanie**



Rysunek 87: Przed - dodawanie  
z stałą



Rysunek 88: Po (bez normaliza-  
cji, wsp=50)



Rysunek 89: Po (z normalizacją,  
wsp=50)



Rysunek 90: Przed - dodawanie dwóch obrazów



Rysunek 91: Przed - dodawanie dwóch obrazów



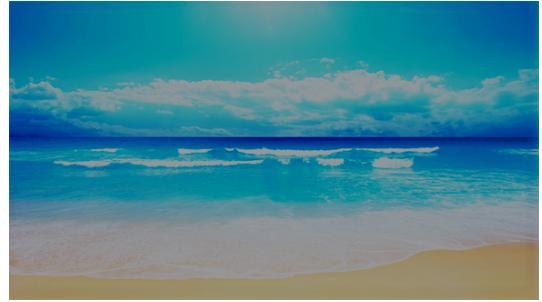
Rysunek 92: Po (bez normalizacji)



Rysunek 93: Po (z normalizacją)



Rysunek 94: Przed - odejmowanie z stałą



Rysunek 95: Po (bez normalizacji, wsp=100)



Rysunek 96: Po (z normalizacją, wsp=100)



Rysunek 97: Przed - odejmowanie dwóch obrazów



Rysunek 98: Przed - odejmowanie dwóch obrazów



Rysunek 99: Po (bez normalizacji)



Rysunek 100: Po (z normalizacją)

## **7.2 Mnożenie obrazu przez zadaną liczbę oraz przez inny obraz**

### **7.2.1 Opis działania**

Rozwiążanie składa się z dwóch osobnych algorytmów.

Pierwszy z nich jest algorytmem mnożącym dwa obrazy barwowe, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje dalsze instrukcje. Jeśli piksel wynosi 255 (maksymalną wartość piksla) to jego wartość zostaje przepisana do tablicy wynikowej. Podobnie jeśli piksel wynosi 0 czyli wartość minimalną. Dla reszty piksli zostaje wykonane mnożenie piksli obrazu pierwszego i drugiego oraz podzielenie ich przez 256. Wartość zostanie zaokrąglona w góre.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.2.1(gotowy).png*.

Drugim algorytmem jest mnożenie obrazu szarego ze stałą. Wszystko odbywa się analogicznie jak wyżej tylko, że dla jednego obrazu. Drugi obraz zastąpiony jest pobraną przez funkcję stałą. Obraz jest zapisywany jako *img3.2.2(gotowy).png*.

### **7.2.2 Ograniczania, uwagi**

-

### **7.2.3 Możliwe problemy**

-

### **7.2.4 Przykładowe wykonanie**



Rysunek 101: Przed



Rysunek 102: Po (bez normalizacji, wsp=70)



Rysunek 103: Po (z normalizacją, wsp=70)



Rysunek 104: Przed



Rysunek 105: Przed



Rysunek 106: Po (bez normalizacji)



Rysunek 107: Po (z normalizacją)

## **7.3 Mieszanie obrazów z określonym współczynnikiem**

### **7.3.1 Opis działania**

Algorytm miesza dwa obrazy barwowe z współczynnikiem, rozpoczyna się od wczytania plików. Pliki są otwierana bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje algorytm:  $wspoczynnik * PikselObrazuPierwszego + (1 - wspoczynnik) * PikselObrazuDrugiego$ . Wartość jest zaokrąglana w góre.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.3(gotowy).png*.

### **7.3.2 Ograniczania, uwagi**

-

### **7.3.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **7.3.4 Przykładowe wykonanie**



Rysunek 108: Przed



Rysunek 109: Przed



Rysunek 110: Po (bez normalizacji, wsp=30)



Rysunek 111: Po (z normalizacją, wsp=30)



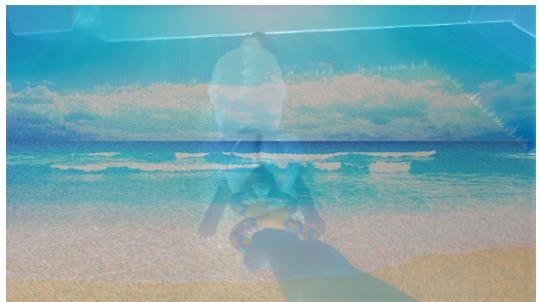
Rysunek 112: Przed



Rysunek 113: Przed



Rysunek 114: Po (bez normalizacji, wsp=10)



Rysunek 115: Po (z normalizacją, wsp=10)

## **7.4 Potęgowanie obrazu z zadaną potęgą**

### **7.4.1 Opis działania**

Algorytm potęguje obraz barwowy podaną potęgą, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest potęgowany do zadanej potęgi ( $\geq 1$ ) a wynik jest zaokrąglany w góre i zapisywany do tablicy wynikowej.

Po tych czynnościami uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.4(gotowy).png*.

### **7.4.2 Ograniczania, uwagi**

-

### **7.4.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą lub stała będzie ujemna. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **7.4.4 Przykładowe wykonanie**



Rysunek 116: Przed



Rysunek 117: Po (bez normalizacji, potęga=2)



Rysunek 118: Po (z normalizacją, potęga=2)



Rysunek 119: Przed

Rysunek 120: Po (bez normalizacji, potęga=3)



Rysunek 121: Po (z normalizacją, potęga=3)

## **7.5 Dzielenie obrazu przez liczbę oraz przez inny obraz**

### **7.5.1 Opis działania**

Rozwiążanie składa się z dwóch osobnych algorytmów.

Pierwszy algorytm dzieli dwa obrazy barwowe, rozpoczyna się od wczytania plików. Pliki są otwierane bitowo, program sprawdza czy są one typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Następnie obrazy są sprawdzane pod kątem ich rozmiaru i większy zostaje gotową funkcją biblioteczną, sprowadzony do rozmiaru mniejszego obrazu.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Dla każdego piksla wynikowego wykonuje kolejne kroki. Wartości piksla pierwszego i drugiego są dodawane oraz w celu zabezpieczenia przed dzieleniem przez 0 dodawana jest do nich wartość 1. Następnie otrzymana suma jest mnożona przez 255 i dzielona przez opisaną wcześniej  $sum + 1$ . Wynik jest zaokrąglany w górę i zapisywany do tablicy wynikowej.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.5.1(gotowy).png*.

Drugi algorytm działa analogicznie do pierwszego z różnicą, że drugi obraz zastępowany jest stałą. Obraz jest zapisywany jako *img3.5.2(gotowy).png*.

### **7.5.2 Ograniczania, uwagi**

-

### **7.5.3 Możliwe problemy**

Program nie wykona się jeśli stała nie będzie liczbą całkowitą. Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **7.5.4 Przykładowe wykonanie**



Rysunek 122: Przed



Rysunek 123: Po (bez normalizacji, wsp=20)



Rysunek 124: Po (z normalizacją, wsp=20)



Rysunek 125: Przed



Rysunek 126: Przed



Rysunek 127: Po (bez normalizacji)



Rysunek 128: Po (z normalizacją)

## **7.6 Pierwiastkowanie obrazu**

### **7.6.1 Opis działania**

Algorytm pierwiastkuje czyli potęguje obraz barwowy podaną potegą, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest potęgowany do zadanej potęgi (z przedziału [0, 1]) a wynik jest zaokrąglany w góre i zapisywany do tablicy wynikowej.

Po tych czynnościami uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.6(gotowy).png*.

### **7.6.2 Ograniczania, uwagi**

Program powinien przyjmować liczby z przedziału [0,1], ale można wpisać też liczby z zakresu [0,4).

### **7.6.3 Możliwe problemy**

Program nie wykona się liczba będzie całkowita lub dziesiętna  $\zeta = 4$  lub  $\mathfrak{j} = 0$ . Należy wrócić na stronę główną i uruchomić jeszcze raz.

### **7.6.4 Przykładowe wykonanie**



Rysunek 129: Przed



Rysunek 130: Po (bez normalizacji, pierw=0.3)



Rysunek 131: Po (z normalizacją, pierw=0.3)



Rysunek 132: Przed



Rysunek 133: Po (bez normalizacji, pierw=0.7)



Rysunek 134: Po (z normalizacją, pierw=0.7)

## **7.7 Logarytmowanie obrazu**

### **7.7.1 Opis działania**

Algorytm logarytmuje obraz barwowy, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następnie każdy piksel jest powiększany o 1 i jest logarytmowany z podstawą logarytmu 10, za pomocą gotowej funkcji matematycznej.

Po tych czynnościach uruchamiana jest napisana odręcznie funkcja normalizacji i obraz jest zapisywany jako *img3.7(gotowy).png*.

### **7.7.2 Ograniczania, uwagi**

-

### **7.7.3 Możliwe problemy**

-

### **7.7.4 Przykładowe wykonanie**



Rysunek 135: Przed



Rysunek 136: Po (bez normalizacji)



Rysunek 137: Po (z normalizacją)



Rysunek 138: Przed



Rysunek 139: Po (bez normalizacji)



Rysunek 140: Po (z normalizacją)

## 8 Operacje geometryczne na obrazie

### 8.1 Przemieszczenie obrazu o zadany wektor

#### 8.1.1 Opis działania

Algorytm przemieszcza obraz o podany wektor czyli dwie liczby ( $x, y$ ), rozpoznaje się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm odwraca współrzędną  $y$  żeby zachować układ punktu (0,0) w lewym dolnym rogu. Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następne instrukcje wykonuje dla każdego piksela obrazu. Sprawdza czy położenie  $y$  piksla plus jego przesunięcie mieści się w rozmiarach obrazu. Do samego sprawdza dla  $x$  piksla i szerokości obrazu. Jeśli warunek jest spełniony to do tablicy pomocniczej przepisuje wartość piksla obrazu początkowego, z indeksem przesuniętym od pobraną wysokość i szerokość.

Po tych czynnościami obraz jest zapisywany jako *img4.1(gotowy).png*.

#### 8.1.2 Ograniczania, uwagi

Obraz przesuwa się, ale nie tworzy dodatkowego pola, tzn. znika w stronę w którą została przesunięty, zostawiając czarne pole. Obraz można przesunąć w każdą stronę tzn. akceptowalne są wartości dodatnie i ujemne. Nie są akceptowalne natomiast wartości dziesiętne. Jeśli podane przesunięcie będzie większe niż rozmiary obrazu to otrzymamy pusty/czarny obraz.

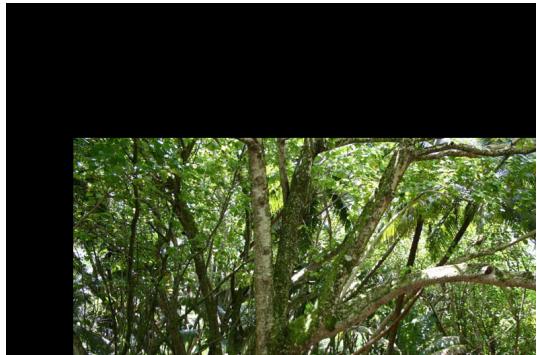
#### 8.1.3 Możliwe problemy

Program nie wykona się jeśli wartość będzie dziesiętna. Należy wrócić do menu i drugi raz włączyć wybraną funkcję.

#### 8.1.4 Przykładowe wykonanie



Rysunek 141: Przed



Rysunek 142: Po (przesunięcie o  
 $100, -200$ )



Rysunek 143: Przed



Rysunek 144: Po ( przesunięcie  
o  $0, 100$ )

## 8.2 Jednorodne i niejednorodne skalowanie obrazu

### 8.2.1 Opis działania

Algorytm skaluje obraz o podany wektor czyli dwa współczynniki ( $x, y$ ), rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu pomnożoną przez podane wartości do skalowania(zaokrąglone w góre). Dla każdego piksla sprawdza:

1. Czy zaokrąglona w góre, wartość poziomego położenia piksla podzielona przez współczynnik x jest większa bądź równa 0, oraz jednocześnie czy jest mniejsza od szerokości obrazu.
2. Czy zaokrąglona w góre, wartość pionowego położenia piksla podzielona przez współczynnik y jest większa bądź równa 0, oraz jednocześnie czy jest mniejsza od wysokości obrazu.

Jeśli oba warunki są spełnione to do tablicy pomocniczej przepisuje wartości obrazu wejściowego, dzieląc ich poziome i pionowe położenie przez odpowiedni współczynnik.

Po tych czynnościach obraz jest zapisywany jako *img4.2(gotowy).png*.

### 8.2.2 Ograniczania, uwagi

Dla ujemnych wartości zostaną one zamienione na dodatnie w celu uniknięcia błędu.

### 8.2.3 Możliwe problemy

-

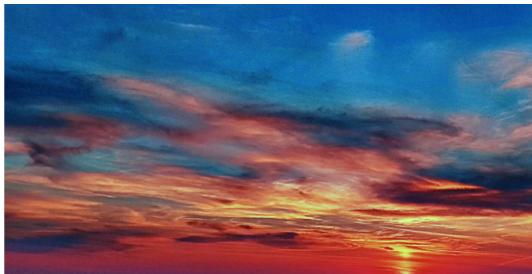
### 8.2.4 Przykładowe wykonanie



Rysunek 145: Przed



Rysunek 146: Po (przesunięcie o 2, 3)



Rysunek 147: Przed



Rysunek 148: Po ( przesunięcie o 2, 0,5)

## 8.3 Obracanie obrazu o dowolny kąt

### 8.3.1 Opis działania

Algorytm obraca obraz o dowolny kąt podany w radianach, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o wielkości wczytanego obrazu. Następne instrukcje wykonuje dla każdego piksela obrazu. Liczy przyszła położenie pionowe i poziome piksla. Dla pionowego położenia mnoży aktualne pionowe położenie razy sinus kąta, dodaje do tego poziome położenie pomnożone przez cosinus kąta. Całość zaokrąglą w góre. Dla poziomego położenia mnoży pionowe położenie piksla przez cosinus kąta i odejmuje poziome położenie pomnożone przez sinus kąta. Całość zaokrąglą w góre. Następnie sprawdza warunek: czy obliczone poziome położenie jest większe od 0 i jednocześnie mniejsze od szerokości obrazu. Jednocześnie musi być też spełniony warunek: czy obliczone pionowe położenie jest większe od 0 i jednocześnie mniejsze od wysokości obrazu. Jeśli oba są spełnione to do tablicy pomocniczej, dla obliczonych nowych położień, przypisuje wartości piksli obrazu wejściowego.

Po zakończeniu przepisywania, wypełnia pola pozostałe, wypełnione wartością -1. Dla każdego piksła sprawdza czy jego wartość wynosi -1. Jeśli tak to sprawdza czy położenie pionowe piksła jest mniejsze od wysokości. Jeśli jest to przepisuje do tablicy pomocniczej wartość piksła z tablicy pomocniczej, znajdującego się pod nim. W przeciwnym przypadku przepisuje piksel znajdujący się pod nim. Następnie sprawdza czy wartość danego piksła nadal wynosi -1, co oznaczałoby, że poprzedni warunek nie został spełniony. Jeśli tak to przypisuje pikselowi wartość 0.

Po tych czynnościach obraz jest zapisywany jako *img4.3(gotowy).png*.

### 8.3.2 Ograniczania, uwagi

Obraz przesuwa się, ale nie tworzy dodatkowego pola, tzn. znika w stronę w której został przesunięty, zostawiając czarne pole. Obraz można przesunąć w każdą stronę tzn. akceptowalne są wartości dodatnie, ujemne i dziesiętne. Podawana wartość kąta ma być w radianach.

Mam wątpliwości co do poprawności zrealizowania zadania tzn. obracane obraz zdają się nie obracać względem punktu (0,0) ustalonego w lewym, dolnym rogu. Obrót o 30 stopni bardziej przypomina obrót o 60 stopni. Algorytm został wzięty z Pana książki, ale tablice w języku programistycznym mają swój początek (0,0) w lewym górnym rogu. Możliwe, że to powoduje błąd. Niemniej

jednak algorytm obraca obraz.

Widać poszarpane końce obrazu, ze względu na użycie mało zaawansowanego algorytmu interpolacji - najbliższy sąsiad.

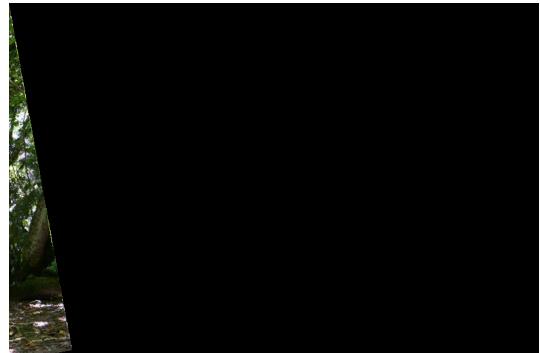
### **8.3.3 Możliwe problemy**

-

### **8.3.4 Przykładowe wykonanie**



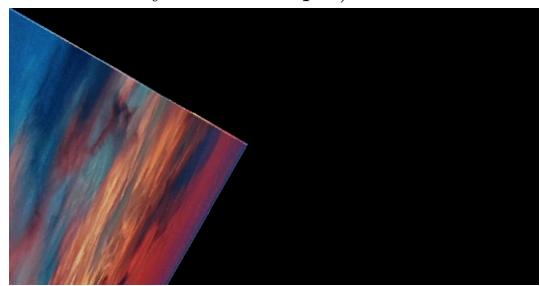
Rysunek 149: Przed



Rysunek 150: Po (kąt w radianach 1.39 czyli ok. 80 stopni)



Rysunek 151: Przed



Rysunek 152: Po (kąt w radianach 0.52 czyli ok. 30 stopni)

## 8.4 Symetrie względem osi układu i zadanej prostej

### 8.4.1 Opis działania

Program składa się w dwóch osobnych algorytmów.

Algorytm pierwszy tworzy symetryczne odbicie względem osi układu, rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program tworzy tablice pomocniczą o rozmiarach dwa razy większych niż aktualny obraz. Następne instrukcje wykonuje dla każdego piksela obrazu. Algorytm liczy nowe położenie dla każdego piksła. Od aktualnego położenia odejmuje szerokość (lub wysokość) a następnie odejmuje wynik od szerokości (lub wysokości). Otrzymane wyniki są nowymi współrzędnymi piksela, dla nich do tablicy pomocniczej przypisywany jest piksel obrazu wejściowego. Obraz jest teraz w prawym dolnym rogu dwa razy większej tablicy, ale obrócony według osi.

Następnie algorytm likwiduje zbędne miejsce obok obrazu. Tworzy kolejną drugą tablice pomocniczą, o wymiarach starego obrazu. Dla każdego piksła podwójnej wysokości i szerokości, wykonuje podane obliczenia. Jeśli wysokość obrazu jest większa od 0 i jednocześnie mniejsza od aktualnej wielkości piksła oraz jeśli szerokość obrazu jest większa od 0 i jednocześnie mniejsza od aktualnej szerokości piksła to do drugiej tablicy pomocniczej przepisujemy obraz z pierwszej tablicy pomocniczej. Robimy to dla indeksów piksła odejmując odpowiednio wysokość lub szerokość.

Po tych czynnościami obraz jest zapisywany jako *img4.4.1(gotowy).png*.

Drugi algorytm tworzy symetryczne odbicie względem podanej prostej pionowej czyli współrzędnej punktu x. Również rozpoczyna od wczytania pliku i sprawdzenia go oraz pobiera wysokość, szerokość i kanały.

Program tworzy tablice pomocniczą o szerokości dwa razy większej niż aktualny obraz i takiej samej wysokości. Dla każdego piksła, dla nowej wielkości tablicy sprawdza następujące warunki. Jeśli indeks szerokości piksła jest większy bądź równy podanej współrzędnej punktu x to do tablicy pomocniczej przepisywana jest wartość piksła z obrazu wynikowego. Z tą różnicą, że indeks odpowiadający za szerokość obrazu wejściowego jest obliczany poprzez podwojenie punktu x linii i odjęcie od niego aktualnego indeksu szerokości. Jeśli natomiast warunek nie został spełniony to przepisuje piksel z obrazu wejściowego do tablicy pomocniczej.

Po tych czynnościami obraz jest zapisywany jako *img4.4.2(gotowy).png*.

#### **8.4.2 Ograniczania, uwagi**

Algorytm drugi tworzy odbicie symetryczne względem linii pionowych. Odbicie względem linii poziomych byłoby tworzone analogiczne, ale uznałem, że nie będę go już implementował.

#### **8.4.3 Możliwe problemy**

Program nie będzie się wykonywał jeśli podana wartość  $x$  linii będzie niecałkowita albo mniejsza od zera.

#### **8.4.4 Przykładowe wykonanie**



Rysunek 153: Przed



Rysunek 154: Po



Rysunek 155: Przed



Rysunek 156: Po



Rysunek 157: Przed



Rysunek 158: Po - linia w punkcie 450



Rysunek 159: Przed



Rysunek 160: Po - linia w punkcie 200

## **8.5 Wycinanie fragmentów obrazu**

### **8.5.1 Opis działania**

Algorytm wycina czarny prostokąt w obrazie, zostawiając tam czarne pole. Robi to na podstawie wprowadzonego (x, y) lewego dolnego punktu i prawego górnego punktu pola, które ma zostać wycięte. Rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program zamienia wprowadzone dane wysokości tak, aby odpowiadały układowy o punkcie (0,0) w lewym dolnym rogu. Algorytm sprawdza czy wymiary któregoś z punktów nie wychodzą poza obraz, jeśli tak to program się wyłącza. Następnie tworzy on tablice pomocniczą o rozmiarach obrazu. Kolejne instrukcje wykonuje dla każdej szerokości i wysokości piksla obrazu wejściowego. Jeśli wartości aktualnej szerokości i wysokości piksla są w podanych wymiarach wycięcia to nic nie robi. W przeciwnym wypadku jeśli współrzędne piksla są poza lub na granicy podanego prostokąta to przepisuje wartość piksla do tablicy pomocniczej.

Po tych czynnościami obraz jest zapisywany jako *img4.5(gotowy).png*.

### **8.5.2 Ograniczania, uwagi**

Dane wprowadzane mogą być nieprawidłowe (większe od obrazu) i wtedy algorytm nie będzie działał, jest to zabezpieczenie, ale nie obsługa błędu.

Jeśli wprowadzone współrzędne punktów będą wskazywały, że punkt drugi będzie tym lewym dolnym a pierwszy tym prawym górnym to nic się nie wytnie.

### **8.5.3 Możliwe problemy**

Program nie będzie się wykonywał jeśli wartość nie będzie całkowita.

### **8.5.4 Przykładowe wykonanie**



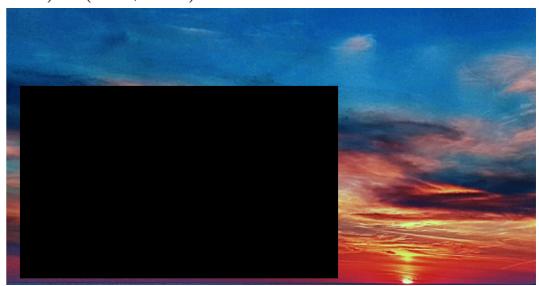
Rysunek 161: Przed



Rysunek 162: Po - punkty (200, 100) i (500, 300)



Rysunek 163: Przed



Rysunek 164: Po - punkty (20, 10) i (500, 300)

## 8.6 Wycinanie fragmentów obrazu

### 8.6.1 Opis działania

Algorytm wycina prostokąt w obrazie i zapisuje jako osobny obraz. Robi to na podstawie wprowadzonego (x, y) lewego dolnego punktu i prawego górnego punktu pola, które ma zostać wycięte. Rozpoczyna się od wczytania pliku. Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Program zamienia wprowadzone dane wysokości tak, aby odpowiadały układowy o punkcie (0,0) w lewym dolnym rogu. Algorytm sprawdza czy wymiary któregoś z punktów nie wychodzą poza obraz, jeśli tak to program się wyłącza. Następnie tworzy on tablice pomocniczą o rozmiarach obrazu. Kolejne instrukcje wykonuje dla każdej szerokości i wysokości piksla obrazu wejściowego. Jeśli wartości aktualnej szerokości i wysokości piksla są w podanych wymiarach wycięcia to zapisuje je do tablicy pomocniczej. Obraz jest w tablicy z czarnymi ramkami.

Po tej operacji tworzy kolejną tablice pomocniczą o odpowiednio zmniejszonej wielkości. Następnie dla każdego piksla wprowadza przepisaną wartość z poprzedniej tablicy pomocniczej, przesuniętą odpowiednio dla podanych punktów. Obraz jest już wycięty, w tablicy o odpowiednim rozmiarze.

Po tych czynnościach obraz jest zapisywany jako *img4.6(gotowy).png*.

### 8.6.2 Ograniczania, uwagi

Dane wprowadzane mogą być nieprawidłowe (większe od obrazu) i wtedy algorytm nie będzie działał, jest to zabezpieczenie, ale nie obsługa błędu.

Jeśli wprowadzone współrzędne punktów będą wskazywały, że punkt drugi będzie tym lewym dolnym a pierwszy tym prawym górnym to nic się nie wytnie.

### 8.6.3 Możliwe problemy

Program wyłączy się jeśli wartość nie będzie całkowita.

Zdjęcie wyjściowe posiada czarny pasek o grubości jednego piksla na dole i z prawej strony.

### 8.6.4 Przykładowe wykonanie



Rysunek 165: Przed



Rysunek 166: Po - punkty (300, 130) i (421, 150)



Rysunek 167: Przed



Rysunek 168: Po - punkty (20, 123) i (300, 200)

## 9 Operacje na histogramie obrazu szarego

W funkcjach tworzących histogramy, są one tworzone za pomocą zliczania piksli i tworzenia kolumny o wysokości równej ilości piksla danej wartości. Taki obraz jest zapisywany do pliku aplikacji, ale nie wyświetlany. Wyświetlany jest za to wykres utworzony funkcją biblioteczną, zrobiony z tych samych danych co wcześniejszy histogram. Jest tak, ponieważ nie umiem zrobić odpowiedniej skali do tworzonego przez mnie histogramu. W dokumentacji załączone są oba histogramy w celu właśnie odniesienia wykresu do reprezentowanych przez niego wartości.

### 9.1 Obliczanie histogramu

#### 9.1.1 Opis działania

Algorytm tworzy histogram obrazu.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocniczą do której pobiera tylko piksle pierwszego kanału obrazu. Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 100 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksla. Dla każdej z 255 wartości sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksla w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksła się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramu o 90 stopni. Zapisuje histogram jako *img5.1(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych.

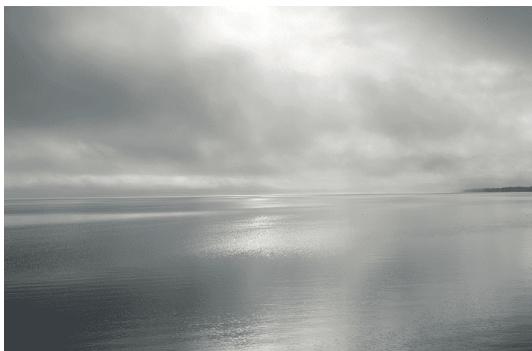
#### 9.1.2 Ograniczania, uwagi

-

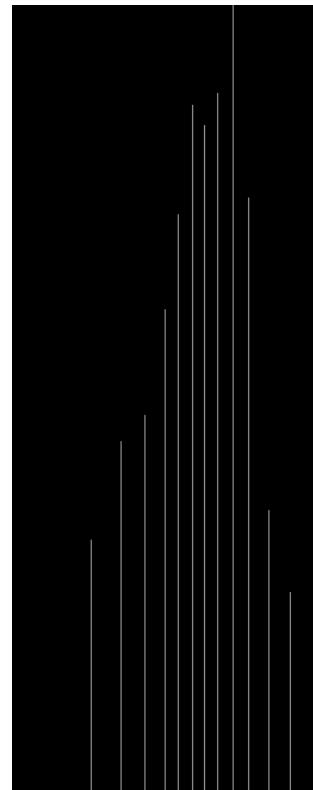
#### 9.1.3 Możliwe problemy

-

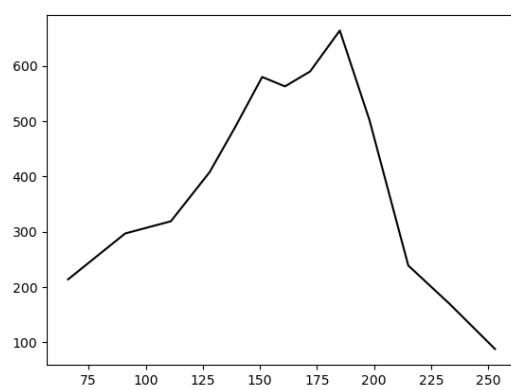
#### 9.1.4 Przykładowe wykonanie



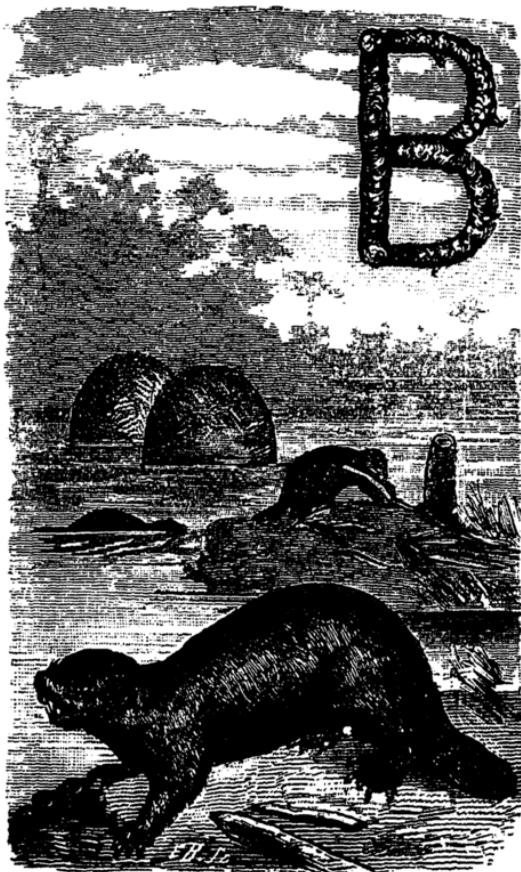
Rysunek 169: Zdjęcie



Rysunek 170: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



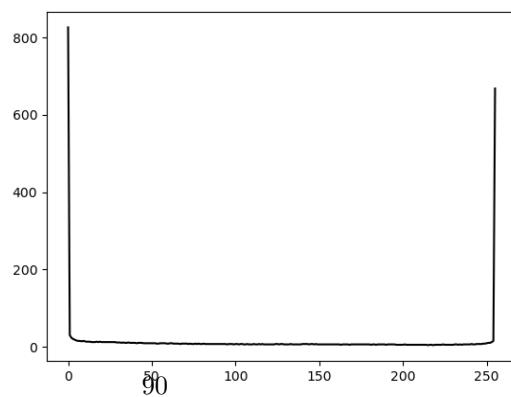
Rysunek 171: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 172: Zdjęcie



Rysunek 173: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 174: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)

## 9.2 Przemieszczanie histogramu

### 9.2.1 Opis działania

Algorytm tworzy histogram obrazu i przemieszcza go.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocniczą do której pobiera tylko piksle pierwszego kanalu obrazu. Dodatkowo, jeśli  $wartoscpiksla + podanawartosc < 0$  to przypisuje danemu pikslowi wartość 0. W przeciwnym przypadku, jeśli  $wartoscpiksla + podanawartosc > 255$  to przypisuje danemu pikslowi wartość 255. W przeciwnym przypadku przypisuje pikslowi  $wartoscpiksla + podanawartosc$ . Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 100 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksła. Dla każdej z 255 wartości sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksła w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksła się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramu o 90 stopni. Zapisuje histogram jako *img5.2(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych. Zapisuje też zmieniony obraz jako *img5.2obr(gotowy).png*

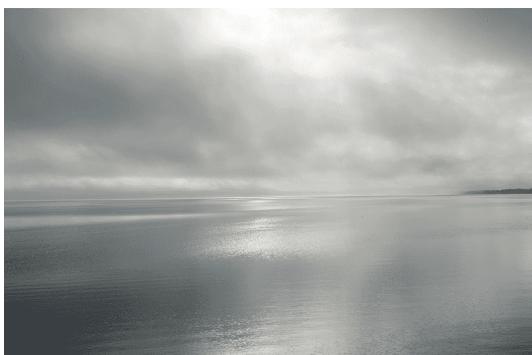
### 9.2.2 Ograniczania, uwagi

Próg powinien być z zakresu [0, 255] oraz być liczbą całkowitą. Dla liczb z poza zakresu algorytm wykona się, ale nie będzie to miało bez sensu.

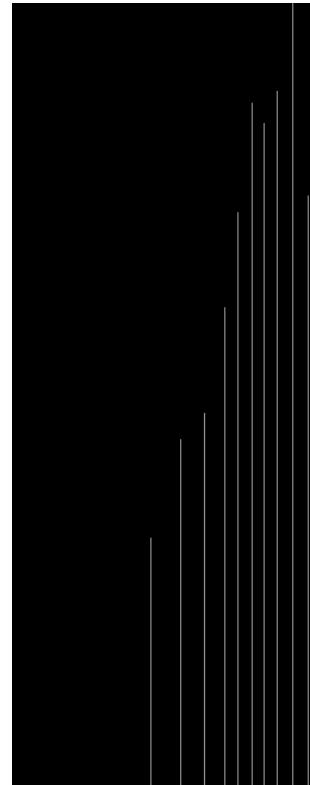
### 9.2.3 Możliwe problemy

Program nie zadziała dla wprowadzonych wartości dziesiętnych.

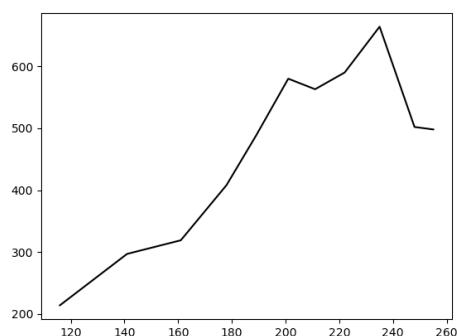
### 9.2.4 Przykładowe wykonanie



Rysunek 175: Zdjęcie przed



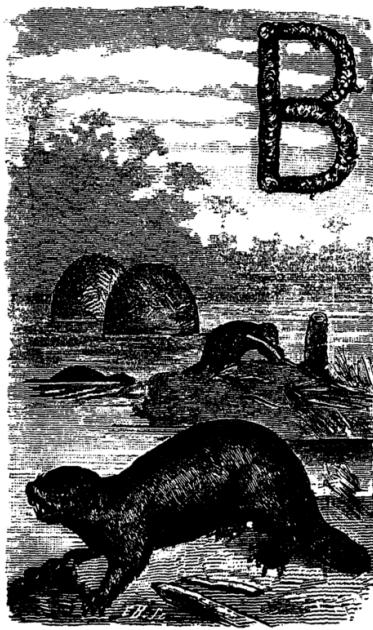
Rysunek 176: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 50)



Rysunek 177: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 50)



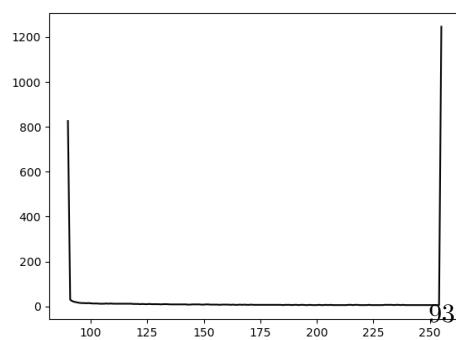
Rysunek 178: Zdjęcie po, wsp = 50



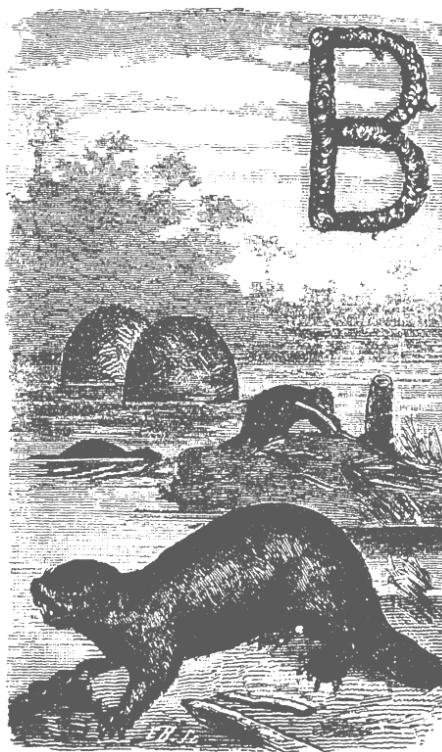
Rysunek 179: Zdjęcie przed



Rysunek 180: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 90)



Rysunek 181: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 90)



Rysunek 182: Zdjęcie po, wsp = 90

## 9.3 Rozciąganie histogramu

### 9.3.1 Opis działania

Algorytm tworzy histogram obrazu i rozciąga go.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocniczą do której pobiera tylko piksle pierwszego kanału obrazu. Wyszukuje minimalną i maksymalną wartość piksla w obrazie. Następnie dla każdego piksla sprawdza, jeśli  $wartosc_piksla < minwartosc$  to przypisuje danemu pikslowi wartość 0. W przeciwnym przypadku, jeśli  $wartosc_piksla > makswartosc$  to przypisuje danemu pikslowi wartość 255. W przeciwnym przypadku przypisuje pikslowi  $((wartosc_piksla - minwartosc) * 255) / (makswartosc - minwartosc)$ . Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 100 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksla. Dla każdej z 255 wartości sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksla w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksła się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramu o 90 stopni. Zapisuje histogram jako *img5.3(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych. Zapisuje też zmieniony obraz jako *img5.3obr(gotowy).png*

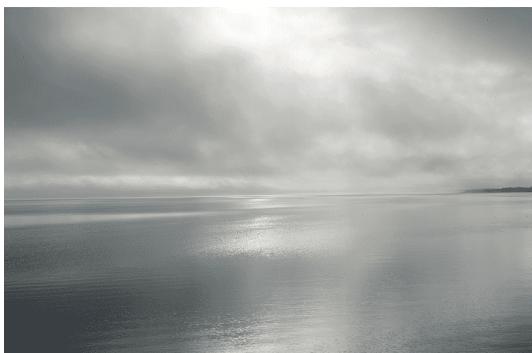
### 9.3.2 Ograniczania, uwagi

-

### 9.3.3 Możliwe problemy

-

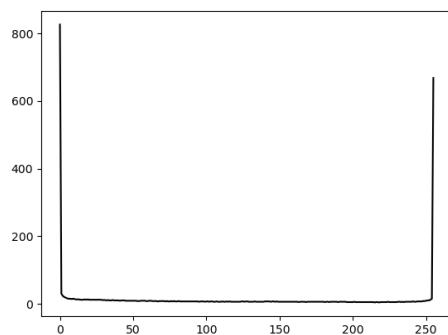
### 9.3.4 Przykładowe wykonanie



Rysunek 183: Zdjęcie przed



Rysunek 184: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 185: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)

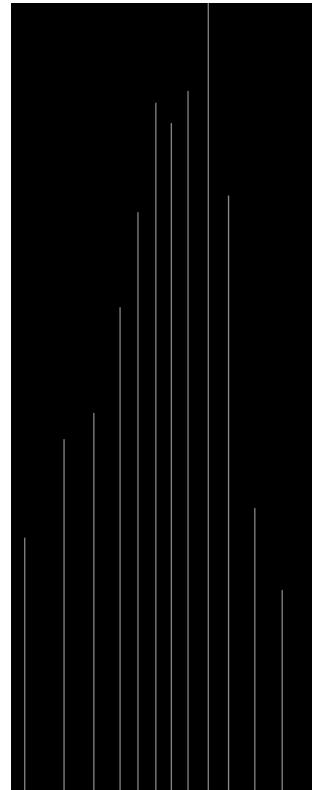
95



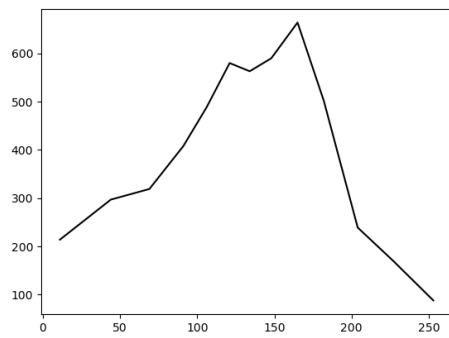
Rysunek 186: Zdjęcie po



Rysunek 187: Zdjęcie przed

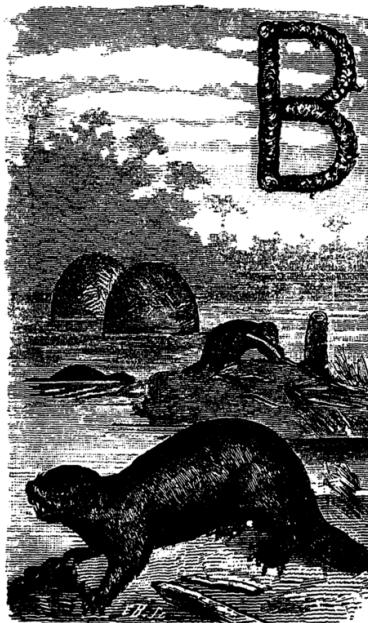


Rysunek 188: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 189: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)

96



Rysunek 190: Zdjęcie po

## **9.4 Progowanie lokalne**

### **9.4.1 Opis działania**

Algorytm znajduje próg lokalny w obrazu (obszar 5 pikseli w każdą stronę) i proguję obraz.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice i przerzuca do niej tylko pierwszy kanał obrazu. Następnie dla każdego piksla: 1. Jeśli piksel znajduje się w środkowej części obrazu (5 pikseli od brzegów obrazu) to pobiera wartość z 5 pikseli z każdej strony od położenia piksla wejściowego. Dzieli tą wartość przez ilość pikseli, to jest próg lokalny. Jeśli wartość piksla jest mniejsza niż próg to wartość zostaje zmieniona na 0 a w przeciwnym przypadku zostaje zmieniona na wartość progu.

2. Jeśli piksel jest w prawej/lewej, pominiętej wcześniej krawędzi obrazu, to zostają wykonane operacje analogiczne do powyższych tylko, że piksle są zbierane tylko z góry i z dołu od piksla wejściowego.
3. Analogicznie do punktu 2, dla pasków na górze i na dole obrazu oraz dla pikseli zbieranych tylko z prawej i lewej strony piksla.

Zapisuje obraz jako *img5.4(gotowy).png*.

### **9.4.2 Ograniczania, uwagi**

W niektórych obrazach mogą być widoczne boczne paski oraz rogi, dla których algorytm jest ograniczony lub nie działa (w przypadku narożników obrazu).

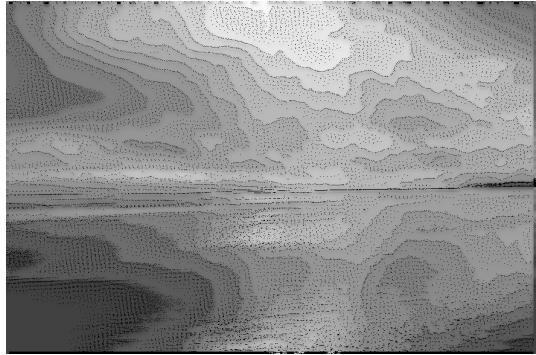
### **9.4.3 Możliwe problemy**

-

### **9.4.4 Przykładowe wykonanie**



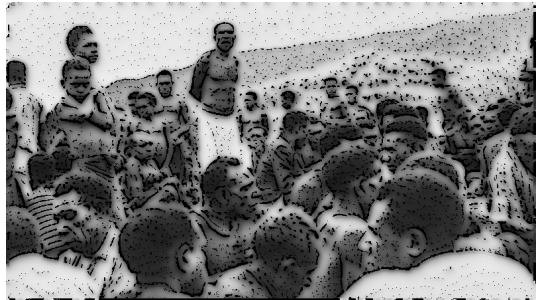
Rysunek 191: Przed



Rysunek 192: Po



Rysunek 193: Przed



Rysunek 194: Po

## **9.5 Progowanie globalne**

### **9.5.1 Opis działania**

Algorytm proguje obraz globalnie, podanym progiem.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice i przerzuca do niej tylko pierwszy kanał obrazu. Następnie dla każdego piksla sprawdza czy jego wartość jest mniejsza od podanego progu. Jeśli jest to zmienia ją na 0, w przeciwnym przypadku zmienia ją na wartość progu.

Zapisuje obraz jako *img5.5(gotowy).png*.

### **9.5.2 Ograniczania, uwagi**

Próg powinien być z zakresu [0, 255] oraz być liczbą całkowitą. Dla liczb z poza zakresu algorytm wykona się, ale nie będzie to miało bez sensu.

### **9.5.3 Możliwe problemy**

Algorytm nie zadziała dla wartości niecałkowitych.

### **9.5.4 Przykładowe wykonanie**



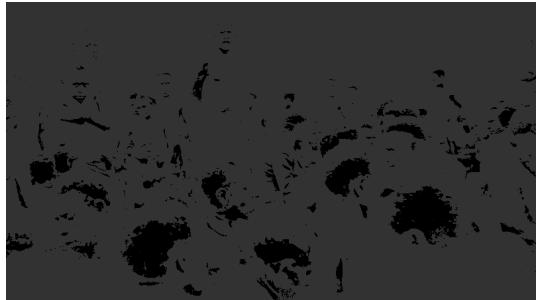
Rysunek 195: Przed



Rysunek 196: Po, wsp = 90



Rysunek 197: Przed



Rysunek 198: Po, wsp = 50

## **10 Operacje na histogramie obrazu barwowego**

W funkcjach tworzących histogramy, są one tworzone za pomocą zliczania piksli i tworzenia kolumny o wysokości równej ilości piksla danej wartości. Taki obraz jest zapisywany do pliku aplikacji, ale nie wyświetlany. Wyświetlany jest za to wykres utworzony funkcją biblioteczną, zrobiony z tych samych danych co wcześniejszy histogram. Jest tak, ponieważ nie umiem zrobić odpowiedniej skali do tworzonego przez mnie histogramu. W dokumentacji załączone są oba histogramy w celu właśnie odniesienia wykresu do reprezentowanych przez niego wartości.

### **10.1 Obliczanie histogramu**

#### **10.1.1 Opis działania**

Algorytm tworzy histogram obrazu.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocnicze do której dzieli obraz na trzy kanały. Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 50 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksła. Dla każdej z 255 wartości, dla każdego kanału sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksla w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksła się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramów o 90 stopni. Łączy je w jedną tablice. Zapisuje histogram jako *img6.1(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych.

#### **10.1.2 Ograniczania, uwagi**

-

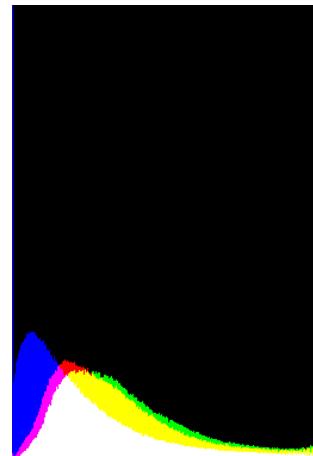
#### **10.1.3 Możliwe problemy**

-

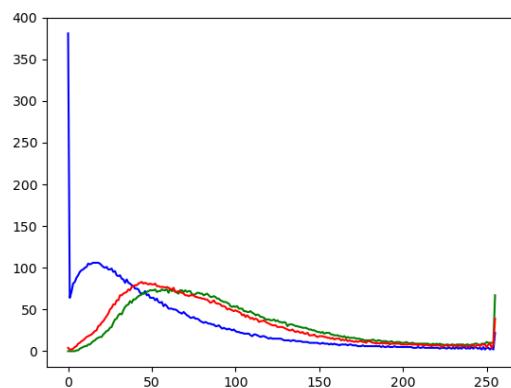
#### **10.1.4 Przykładowe wykonanie**



Rysunek 199: Zdjęcie



Rysunek 200: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



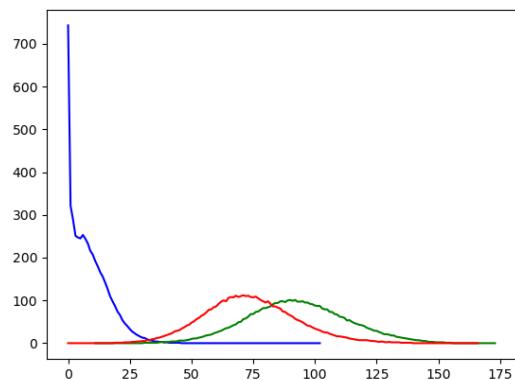
Rysunek 201: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 202: Zdjęcie



Rysunek 203: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 204: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)

## 10.2 Przemieszczanie histogramu

### 10.2.1 Opis działania

Algorytm tworzy histogram obrazu i przemieszcza go.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocnicze do której dzieli obraz na trzy kanały. Dodatkowo, jeśli  $wartosc piksla + podanawartosc < 0$  to przypisuje danemu pikselowi wartość 0. W przeciwnym przypadku, jeśli  $wartosc piksla + podanawartosc > 255$  to przypisuje danemu pikselowi wartość 255. W przeciwnym przypadku przypisuje pikselowi  $wartosc piksla + podanawartosc$ . Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 50 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksla. Dla każdej z 255 wartości, dla każdego kanału sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksela w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksła się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramu o 90 stopni i łączy tablice. Zapisuje histogram jako *img6.2(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych.

### 10.2.2 Ograniczania, uwagi

Próg powinien być z zakresu [0, 255] oraz być liczbą całkowitą. Dla liczb z poza zakresu algorytm wykona się, ale nie będzie to miało bez sensu.

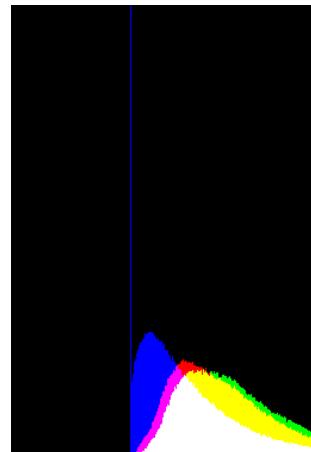
### 10.2.3 Możliwe problemy

Program nie zadziała dla wprowadzonych wartości dziesiętnych.

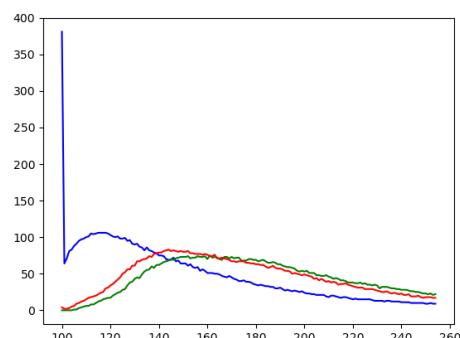
### 10.2.4 Przykładowe wykonanie



Rysunek 205: Zdjęcie przed



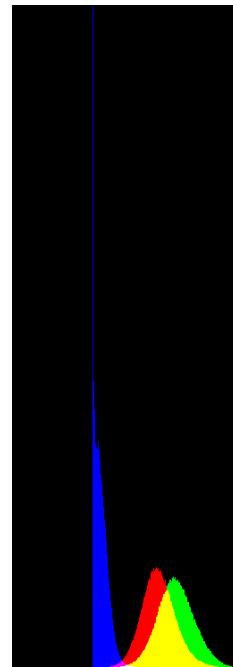
Rysunek 206: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 100)



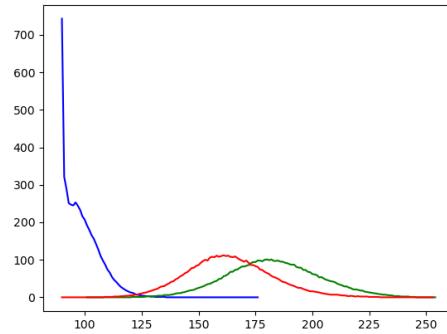
Rysunek 207: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 100)



Rysunek 208: Zdjęcie przed



Rysunek 209: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 90)



Rysunek 210: Histogram (oś y - ilość wystąpień, oś x - wartość piksla, wsp = 90)

## 10.3 Rozciąganie histogramu

### 10.3.1 Opis działania

Algorytm tworzy histogram obrazu i rozciąga go.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm wyszukuje minimalna i maksymalną wartość piksla w obrazie. Następnie dla każdego piksla w sprawdza, jeśli  $wartosc_piksla < minwartosc$  to przypisuje danemu pikslowi wartość 0. W przeciwnym przypadku, jeśli  $wartosc_piksla > makswartosc$  to przypisuje danemu pikslowi wartość 255. W przeciwnym przypadku przypisuje pikslowi  $((wartosc_piksla - minwartosc) * 255) / (makswartosc - minwartosc)$ . Algorytm tworzy tablice pomocnicze do której dzieli kanały obrazu. Następnie za pomocą funkcji bibliotecznej liczy unikatowe wartości piksli oraz ilość ich wystąpień. Po tym dzieli ilość wystąpień na 50 w celu lepszego wyglądu histogramu. Tworzy tablice wynikową o rozmiarze 255 oraz maksymalnej ilości wystąpień jakiejś wartości piksla. Dla każdej z 255 wartości, dla każdego kanału sprawdza czy występuje ona w tym obrazie. Jeśli tak to przypisuje do niej wartość 255 w ilości wystąpień danego piksla w obrazie. Jeśli dana wartość nie występuje to zapisuje do niej 0. Jeśli wartości piksli się skończyły to kończy działanie. Następnie za pomocą gotowej funkcji obraca tablice histogramu o 90 stopni i łączy tablice. Zapisuje histogram jako *img6.3(gotowy).png* oraz tworzy i wyświetla wykres histogramu z wcześniejszych danych.

### 10.3.2 Ograniczania, uwagi

-

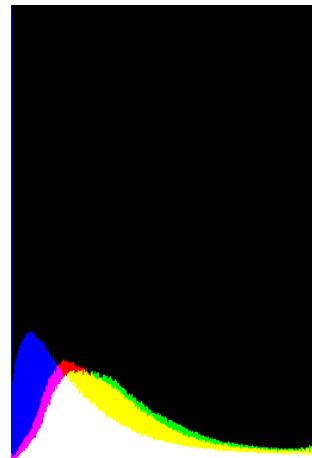
### 10.3.3 Możliwe problemy

-

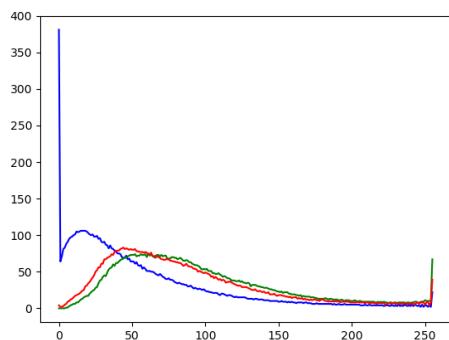
### 10.3.4 Przykładowe wykonanie



Rysunek 211: Zdjęcie przed



Rysunek 212: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



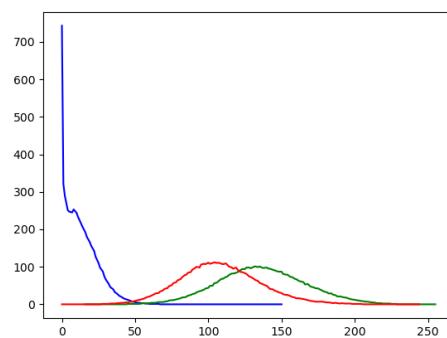
Rysunek 213: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 214: Zdjęcie przed



Rysunek 215: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)



Rysunek 216: Histogram (oś y - ilość wystąpień, oś x - wartość piksla)

## **10.4 Progowanie globalne wieloprogowe**

### **10.4.1 Opis działania**

Algorytm proguje obraz globalnie, podanym trzema progami.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocniczą. Następnie dla każdego piksla sprawdza czy jego wartość jest mniejsza od podanych progów. Jeśli jest mniejsza od progu pierwszego to piksel otrzymuje wartość 0. W przeciwnym przypadku, jeśli jest mniejsza od drugiego progu to piksel otrzymuje wartość progu pierwszego. W przeciwnym przypadku, jeśli jest mniejsza od progu trzeciego to piksel otrzymuje wartość progu drugiego. W przeciwnym przypadku piksel otrzymuje wartość progu trzeciego.

Zapisuje obraz jako *img6.4(gotowy).png*.

### **10.4.2 Ograniczania, uwagi**

Progi powinny być wprowadzona od najmniejszego do największego.

### **10.4.3 Możliwe problemy**

-

### **10.4.4 Przykładowe wykonanie**



Rysunek 217: Przed



Rysunek 218: Po, progi 100,  
150, 200



Rysunek 219: Przed



Rysunek 220: Po, progi 50, 100,  
150

## 10.5 Progowanie lokalne wieloprogowe

### 10.5.1 Opis działania

Algorytm znajduje trzy progi lokalne obrazu (obszar 5 pikseli w każdej stronie) i progguje obraz.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice pomocniczą do której zapisuje wynikowe wartości pikseli. Następnie dla każdego piksla: 1. Jeśli piksel znajduje się w środkowej części obrazu (5 pikseli od brzegów obrazu) to pobiera wartość z 5 pikseli z każdej strony od położenia piksla wejściowego. Dzieli tą wartość przez ilość pikseli, to jest pierwszy próg lokalny. Drugi próg jest dodatkowo mnożony przez 2 a trzecie przez 3. Jeśli jest mniejsza od progu pierwszego to piksel otrzymuje wartość 0. W przeciwnym przypadku, jeśli jest mniejsza od drugiego progu to piksel otrzymuje wartość progu pierwszego. W przeciwnym przypadku, jeśli jest mniejsza od progu trzeciego to piksel otrzymuje wartość progu drugiego. W przeciwnym przypadku piksel otrzymuje wartość progu trzeciego.

2. Jeśli piksel jest w prawej/lewej, pominiętej wcześniej krawędzi obrazu, to zostają wykonane operacje analogiczne do powyższych tylko, że piksle są zbierane tylko z góry i z dołu od piksla wejściowego.

3. Analogicznie do punktu 2, dla pasków na górze i na dole obrazu oraz dla pikseli zbieranych tylko z prawej i lewej strony piksla.

Zapisuje obraz jako *img6.5(gotowy).png*.

### 10.5.2 Ograniczania, uwagi

Progi powinny być z zakresu [0, 255] oraz być liczbami całkowitymi. Dla liczb z poza zakresu algorytm wykona się, ale nie będzie to miało bez sensu.

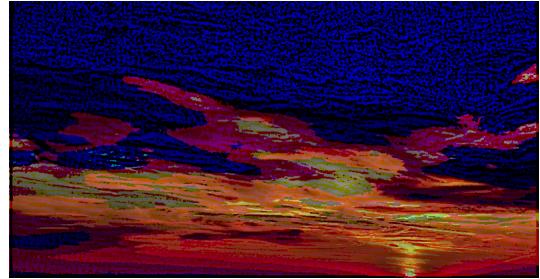
### 10.5.3 Możliwe problemy

Algorytm nie zadziała dla wartości niecałkowitych.

### 10.5.4 Przykładowe wykonanie



Rysunek 221: Przed



Rysunek 222: Po



Rysunek 223: Przed



Rysunek 224: Po

## **10.6 Progowanie lokalne jednoprogowe**

### **10.6.1 Opis działania**

Algorytm znajduje próg lokalny w obrazu (obszar 5 pikseli w każdą stronę) i proguję obraz.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice wynikową. Następnie dla każdego piksla: 1. Jeśli piksel znajduje się w środkowej części obrazu (5 piksli od brzegów obrazu) to pobiera wartość z 5 piksli z każdej strony od położenia piksla wejściowego. Dzieli tą wartość przez liczbę piksli, to jest próg lokalny. Jeśli wartość piksla jest mniejsza niż próg to wartość zostaje zmieniona na 0 a w przeciwnym przypadku zostaje zmieniona na wartość progu.

2. Jeśli piksel jest w prawej/lewej, pominiętej wcześniej krawędzi obrazu, to zostają wykonane operacje analogiczne do powyższych tylko, że piksle są zbierane tylko z góry i z dołu od piksla wejściowego.
3. Analogicznie do punktu 2, dla pasków na górze i na dole obrazu oraz dla piksli zbieranych tylko z prawej i lewej strony piksla.

Zapisuje obraz jako *img5.6(gotowy).png*.

### **10.6.2 Ograniczania, uwagi**

W niektórych obrazach mogą być widoczne boczne paski oraz rogi, dla których algorytm jest ograniczony lub nie działa (w przypadku narożników obrazu).

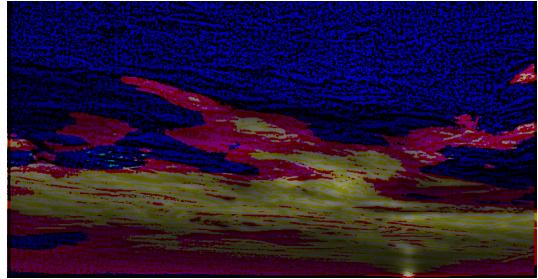
### **10.6.3 Możliwe problemy**

-

### **10.6.4 Przykładowe wykonanie**



Rysunek 225: Przed



Rysunek 226: Po



Rysunek 227: Przed



Rysunek 228: Po

## **10.7 Progowanie globalne jednoprogowe**

### **10.7.1 Opis działania**

Algorytm proguje obraz globalnie, podanym progiem.

Plik jest otwierany bitowo, program sprawdza czy jest on typu PNG czyli czy 2,3,4 znak odpowiadają literą PNG oraz czy pierwszy łańcuch bitów jest opisyany jako IHDR czyli czy 13, 14, 15, 16 znak odpowiadają literą IHDR. Jeśli nie to program się zamyka. Na podstawie kolejnych bitów(16-24) odczytywana jest wysokość i szerokość obrazu. Z wczytanego obrazu odczytywana jest ilość kanałów.

Algorytm tworzy tablice wynikową. Następnie dla każdego piksla sprawdza czy jego wartość jest mniejsza od podanego progu. Jeśli jest to zmienia ją na 0, w przeciwnym przypadku zmienia ją na wartość progu.

Zapisuje obraz jako *img6.7(gotowy).png*.

### **10.7.2 Ograniczania, uwagi**

Próg powinien być z zakresu [0, 255] oraz być liczbą całkowitą. Dla liczb z poza zakresu algorytm wykona się, ale nie będzie to miało bez sensu.

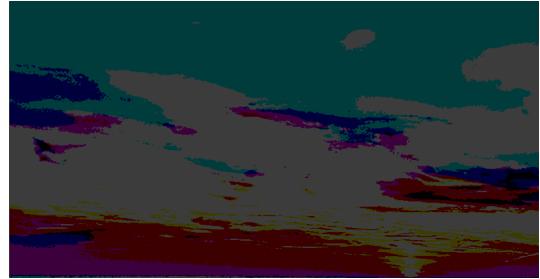
### **10.7.3 Możliwe problemy**

Algorytm nie zadziała dla wartości niecałkowitych.

### **10.7.4 Przykładowe wykonanie**



Rysunek 229: Przed



Rysunek 230: Po, wsp = 60



Rysunek 231: Przed



Rysunek 232: Po, wsp = 110