

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie	Robert Gaca 229 528 Gr. 1	Wydział : WFiIS Rok : I rok II stopień Rok Akademicki : 2014/2015 Termin : Środa
<b>Analiza i przetwarzanie obrazów</b>		
Data wykonania :	Laboratorium nr 3 oraz 4	
Data oddania:	<b>Metody szkieletyzacji w zastosowaniu do liter</b>  <b>Algorytm OCR - segmentacja tekstów</b>	

## I. Metody szkieletyzacji w zastosowaniu do liter.

- **Szkieletyzacja** - wydobycie szkieletu obiektu na zbinaryzowanym obrazie.

- **Szkieletyzacja przez maskę** - potrzebna maska:

x	0	x
x	1	x
1	1	1

- 1 oznacza czarny piksel, 0 oznacza biały piksel, to co stoi na miejscu x jest nieistotne. W celu poprawnego działania algorytmu należy obramować oryginalny obraz ramką o grubości 1 px złożoną z białych pikseli.

### Przebieg algorytmu:

- Każdy czarny piksel (wartość 1) oryginalnego obrazka przyrównujemy do maski oraz jej obrotów o 90, 180 i 270 stopni.
- Jeśli otoczenie piksela spełni którąś z masek, do kopii wstawiana jest wartość 0 (biały), jeśli zaś otoczenie piksela nie spełni żadnej z masek, wstawiamy do kopii wartość 1 (czarny)
- Zastępujemy oryginał kopią.
- Czynność powtarzamy tak długo, jak długo w obrazie zachodzą zmiany.
- **Szkieletyzacja algorytmem KMM** - potrzebne maski:

czwórki = {3, 6, 7, 12, 14, 15, 24, 28, 30, 48, 56, 60, 96, 112, 120, 129, 131, 135, 192, 193, 195, 224, 225, 240}

wycięcia = {3, 5, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63, 65, 67, 69, 71, 77, 79, 80, 81, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 97, 99, 101, 103, 109, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 131, 133, 135, 141, 143, 149, 151, 157, 159, 181, 183, 189, 191, 192, 193, 195, 197, 199, 205, 207, 208, 209, 211, 212, 213, 214, 215, 216, 217, 219, 220, 221, 222, 223, 224, 225, 227, 229, 231, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255}

sprawdzarka = {{128, 64, 32}, {1, 0, 16}, {2, 4, 8}}

Maskę *sprawdzarka* zakłada się na zasadzie sumy wartości. Na początku analizy waga piksela środkowego maski wynosi 0. Jeżeli danej komórce maski odpowiada piksel o wartości 1, 2, 3 lub 4,

to do wagi piksela środkowego dodaje się wagę analizowanej komórki (nie jest ona wymnażana razy wartość piksela). W celu poprawnego działania algorytmu należy obramować oryginalny obraz ramką o grubości 1 px złożoną z białych pikseli.

#### Przebieg algorytmu:

- Wszystkie czarne piksele oznaczamy jako 1, zaś białe jako 0.
- Piksele 1, które posiadają sąsiadów o oznaczeniu 0 po bokach, u góry lub u dołu, oznaczamy jako 2.
- Pozostałe piksele 1, które posiadają sąsiadów o oznaczeniu 0 na rogach, oznaczamy jako 3.
- Dla pikseli oznaczonych jako 2 za pomocą maski *sprawdzarka* obliczamy ich wagę. Jeśli waga znajduje się na liście *czwórki* oznaczenie piksela zamieniamy z 2 na 4.
- Dla wszystkich pikseli oznaczonych jako 4 wyliczamy wagę za pomocą maski *sprawdzarka*. Jeśli waga znajduje się na liście *wycięcia*, zamieniamy piksel na 0, zaś w przeciwnym razie zamieniamy go na 1.
- Dla wszystkich pikseli oznaczonych jako 2 wyliczamy wagę za pomocą maski *sprawdzarka*. Jeśli waga znajduje się na liście *wycięcia*, zamieniamy piksel na 0, zaś w przeciwnym razie zamieniamy go na 1.
- Dla wszystkich pikseli oznaczonych jako 3 wyliczamy wagę za pomocą maski *sprawdzarka*. Jeśli waga znajduje się na liście *wycięcia*, zamieniamy piksel na 0, zaś w przeciwnym razie zamieniamy go na 1.
- Procedurę powtarzamy dopóki w jej trakcie na obrazie zachodzą zmiany.

Dokładny opis KMM.

#### • Szkieletyzacja algorytmem K3M

##### Potrzebne maski:

- Jest to tablica kolejnych masek, w opisie algorytmu kolejne wiersze tablicy będą oznaczane od A0 do A5.

```
private static final int[][] A = {
    {3, 6, 7, 12, 14, 15, 24, 28, 30, 31, 48, 56, 60, 62, 63, 96, 112, 120, 124, 126, 127, 129, 131, 135, 143, 159, 191,
     192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254},
    {7, 14, 28, 56, 112, 131, 193, 224},
    {7, 14, 15, 28, 30, 56, 60, 112, 120, 131, 135, 193, 195, 224, 225, 240},
    {7, 14, 15, 28, 30, 31, 56, 60, 62, 112, 120, 124, 131, 135, 143, 193, 195, 199, 224, 225, 227, 240, 241, 248},
    {7, 14, 15, 28, 30, 31, 56, 60, 62, 63, 112, 120, 124, 126, 131, 135, 143, 159, 193, 195, 199, 207, 224, 225,
     227, 231, 240, 241, 243, 248, 249, 252},
    {7, 14, 15, 28, 30, 31, 56, 60, 62, 63, 112, 120, 124, 126, 131, 135, 143, 159, 191, 193, 195, 199, 207, 224,
     225, 227, 231, 239, 240, 241, 243, 248, 249, 251, 252, 254}
};
```

- Maska jest używana na końcu do likwidacji nadmiarowych sąsiadów.

```
private static final int[] A1pix = {3, 6, 7, 12, 14, 15, 24, 28, 30, 31, 48, 56, 60, 62, 63, 96, 112, 120, 124, 126,
127, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247,
248, 249, 251, 252, 253, 254};
```

- Maska jest używana do wyliczania wagi piksela w celu porównywania jej z wagami z tablic A oraz A1pix. Wagę wylicza się sumując wartości maski dla czarnych lub czerwonych sąsiadów piksela.

```
private static int[][] maska = {{128, 64, 32}, {1, 0, 16}, {2, 4, 8}};
```

#### • Przebieg algorytmu:

- Wszystkie czarne piksele, które spełniają maskę A0 są oznaczane jako graniczne (na przykład na czerwono).

- Wszystkie piksele graniczne, których waga jest w tablicy A1 są usuwane.
- Wszystkie piksele graniczne, których waga jest w tablicy A2 są usuwane.
- Wszystkie piksele graniczne, których waga jest w tablicy A3 są usuwane.
- Wszystkie piksele graniczne, których waga jest w tablicy A4 są usuwane.
- Wszystkie piksele graniczne, których waga jest w tablicy A5 są usuwane.
- Pozostałe piksele graniczne są z powrotem zamieniane na czarne.
- Jeśli podczas powyższych faz na obrazie zaszły zmiany - następuje powrót do fazy szukania granic.
- W przeciwnym razie obraz przeszukiwany jest ostatni raz, tym razem usuwane są czarne piksele, których waga pasuje do maski A1pix.

## II. OCR - segmentacja tekstów

---

- **OCR** (Optical Character Recognition) - jest to rozpoznawanie pisma, tak drukowanego, jak i odręcznego.

Przykładowy obraz pisma do analizy:

Ala ma kota.  
Ala ma kota.  
Ala ma kota.

- **Metoda 1:**

1. **Wyodrębnianie wierszy:**

- Obraz analizowany jest od góry wierszami.
- Jeśli dany wiersz zawiera tylko białe piksele, jest ignorowany.
- Jeśli dany wiersz zawiera czarne piksele, jego pozycja jest zapamiętywana.
- Analizowane są kolejne wiersze, aż znów algorytm trafi na wiersz z samymi białymi pikselami.
- Wszystkie wiersze zawierające czarne piksele, które zostały dotąd znalezione, są kopiowane do nowego obrazka.
- W ten sposób uzyskuje się pierwszy wiersz tekstu.
- Obraz analizowany jest dalej w ten sposób do momentu, aż analiza dojdzie do końca obrazu i wyodrębnione zostaną wszystkie wiersze.

Przykładowy obraz po wyodrębnieniu wierszy (na żółto zaznaczone są linie obrazu, które podlegają wycięciu):

1 Ala ma kota.  
2 Ala ma kota.  
3 Ala ma kota.

2. **Wyodrębnianie liter:**

- Każdy wyodrębniony wiersz analizowany jest kolumnami.
- Jeśli dana kolumna zawiera tylko białe piksele, jest ignorowana.
- Jeśli dana kolumna zawiera czarne piksele, jej pozycja jest zapamiętywana.
- Analizowane są kolejne kolumny, aż znów algorytm trafi na kolumnę z samymi białymi pikselami.

- Wszystkie kolumny zawierające czarne piksele, które zostały dotąd znalezione, są wycinane do nowego obrazka.
- Uzyskany nowy obraz litery analizowany jest wierszami, aż zostanie znaleziony pierwszy czarny piksel, najpierw od góry, a potem od dołu.
- Jeśli od góry lub od dołu znalezione zostaną wiersze bez czarnych pikseli, są one usuwane z obrazka litery.
- W ten sposób uzyskuje się pierwszą literę tekstu.
- Obraz analizowany jest dalej w ten sposób do momentu, aż analiza dojdzie do końca obrazu i wyodrębnione zostaną wszystkie litery dla wszystkich wierszy.

Przykładowy wiersz obrazu po wyodrębnieniu liter (na żółto zaznaczone są kolumny obrazu, które podlegają wycięciu, na czerwono zaznaczone są wiersze liter, które ulegają wycięciu):



- **Metoda 2:**

- Obraz analizowany jest od góry wierszami, do momentu, w którym znaleziony zostaje pierwszy czarny piksel.
- Znaleziony czarny piksel dodawany jest do listy pikseli do przeanalizowania i zaznacza się go innym kolorem, na przykład na zielono.
- Rozpoczynana jest pętla:
  - Dopóki lista czarnych pikseli zawiera elementy
  - Zdejmij pierwszy czarny piksel z listy
  - Czarnych sąsiadów zdjętego piksela dodaj do listy i zaznacz na zielono
- Podczas analizy należy zapamiętać współrzędne skrajnych czarnych pikseli analizowanego obszaru.
- Gdy lista będzie pusta, należy skopiować zielone piksele z analizowanego obszaru do nowego obrazka, w ten sposób wyodrębniając literę. Kopiowane piksele należy zamalować na białe.
- Po wyodrębnieniu litery analizuje się dalej obraz rozpoczynając od piksela obok ostatnio znalezionej czarnego piksela.

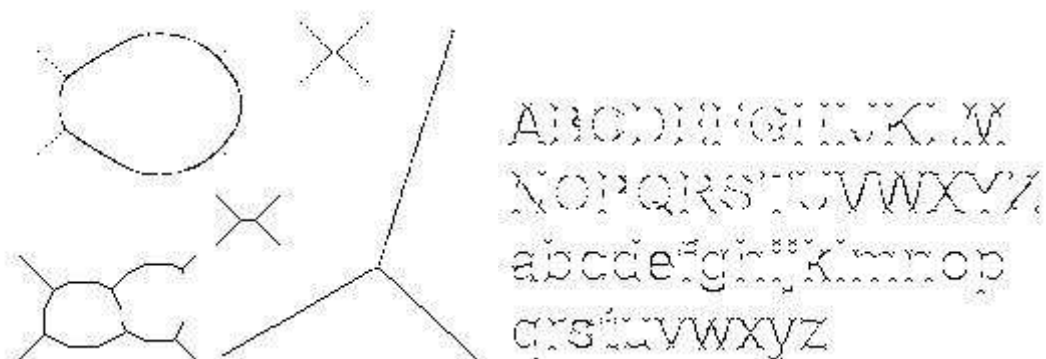
## Ad I. Szkieletyzacja

Obrazy przetwarzane przez wszystkie algorytmy :



### Szkieletyzacja przez maskę :

Wyniki szkieletyzacji na wejściowych obrazach :



## Kod programu :

```
public static BufferedImage Maska(BufferedImage in, int... param) {

    long start_time = System.currentTimeMillis();
    BufferedImage out = new BufferedImage(in.getWidth(), in.getHeight(), in.getType());
    int width;
    int height;

    width = in.getWidth();
    height = in.getHeight();

    in = RGB.powiekszBialymi(in, 1);
    out = RGB.powiekszKopiujac(in, 0);

    int waga = 0;
    int straznik = 0;
    int licznik = 0;

    boolean zmiany;

    do {
        zmiany = false;

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {

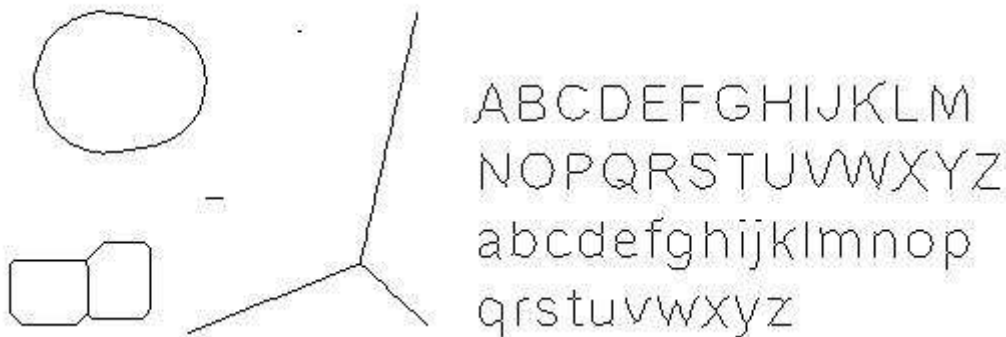
                if (RGB.getR(in.getRGB(i, j)) == 0) {

                    int a[][] = { {RGB.getB(in.getRGB(i-1, j-1))/255, RGB.getB(in.getRGB(i, j-1))/255, RGB.getB(in.getRGB(i+1, j-1))/255},
                                   {RGB.getB(in.getRGB(i-1, j))/255, RGB.getB(in.getRGB(i, j))/255, RGB.getB(in.getRGB(i+1, j))/255},
                                   {RGB.getB(in.getRGB(i-1, j+1))/255, RGB.getB(in.getRGB(i, j+1))/255, RGB.getB(in.getRGB(i+1, j+1))/255}
                                };

                    for (int index = 0; index < 4; index++) {
                        waga = 0;
                        for (int k = 0; k < 3; k++) {
                            for (int l = 0; l < 3; l++) {
                                if (K[index][k][l] == Math.abs(a[k][l] - 1))
                                    waga++;
                            }
                        }
                        if (waga == 5) {
                            out.setRGB(i, j, bialy);
                            zmiany = true;
                        }
                    }
                }
            }
        }
        in = RGB.powiekszKopiujac(out, 0);
    } while (zmiany == true);

    long end_time = System.currentTimeMillis();
    System.out.println("Wykonanie w : " + (end_time - start_time) + "ms");
    return out;
}
```

## Szkieletyzacja algorytmem KMM



## Kod programu :

```
public static BufferedImage KMM ( BufferedImage in){
    long start_time = System.currentTimeMillis();

    BufferedImage out      = new BufferedImage( in.getWidth(), in.getHeight(), in.getType());
    BufferedImage outtmp   = new BufferedImage( in.getWidth(), in.getHeight(), in.getType());
    BufferedImage outbin   = new BufferedImage( in.getWidth(), in.getHeight(), in.getType());

    int width;
    int height;

    width      = in.getWidth();
    height     = in.getHeight();

    in      = RGB.powiekszBialymi(in,1);
    out     = RGB.powiekszKopiujac(in,0);
    outtmp  = RGB.powiekszKopiujac(in,0);
    outbin  = RGB.powiekszKopiujac(in,0);

    int waga      = 0;
    boolean zmiany = false;
    int straznik  = 0;
    int rozmiarMaski = 3;
    int r,g,b;
    int [][] a     = new int[rozmiarMaski][rozmiarMaski];

    do{

        zmiany = false;
        out = RGB.powiekszKopiujac(outtmp,0);
        for (int i = 1; i < width-1; i++) {
            for (int j = 1; j < height-1; j++) {
                if (RGB.getR(out.getRGB(i, j))==0){
                    //krok 1
                    //tworzenie dwuwymiarowej tablicy wartosci koloru sasiadow o wielkosc rozmiarMaski x rozmiarMaski
                    //maska binarna 1 - czarny 0 - bialy
                    for(int q=0;q<rozmiarMaski;q++){
                        for(int w=0;w<rozmiarMaski;w++){
                            a[q][w] = RGB.binar(RGB.getR(out.getRGB(i-rozmiarMaski/2+q,j-rozmiarMaski/2+w)));
                        }
                    }

                    //krok 2 oznaczanie pikseli w pierwszej kolejnosci
                    //jezeli sasiadzi po bokach gora dol to zielony maska = 2
                    if(a[0][1]==0 || a[2][1]==0 || a[1][0]==0 || a[1][2]==0)
                        outtmp.setRGB(i, j, zielony);

                }
            }
        }
        out = RGB.powiekszKopiujac(outtmp,0);
        for (int i = 1; i < width-1; i++) {
            for (int j = 1; j < height-1; j++) {
                if (RGB.getG(out.getRGB(i, j))==0){

                    for(int q=0;q<rozmiarMaski;q++){
                        for(int w=0;w<rozmiarMaski;w++){
                            a[q][w] = RGB.binar(RGB.getR(out.getRGB(i-rozmiarMaski/2+q,j-rozmiarMaski/2+w)));
                        }
                    }

                    //krok 2 oznaczanie pikseli w pierwszej kolejnosci
                    //jezeli corners to niebieski maska = 4
                    if(a[0][0]==0 || a[0][2]==0 || a[2][0]==0 || a[2][2]==0)
                        outtmp.setRGB(i, j, niebieski);

                }
            }
        }

        out = RGB.powiekszKopiujac(outtmp,0);
        for (int i = 1; i < width-1; i++) {
            for (int j = 1; j < height-1; j++) {
                // Wszystkie zielone piksele
                if (RGB.getR(out.getRGB(i, j))==0 && RGB.getG(out.getRGB(i, j))==255 && RGB.getB(out.getRGB(i, j))==0){
                    waga = 0;
                    for(int q=0;q<rozmiarMaski;q++){
                        for(int w=0;w<rozmiarMaski;w++){
                            a[q][w] = RGB.binar(RGB.getR(outbin.getRGB(i-rozmiarMaski/2+q,j-rozmiarMaski/2+w)));
                        }
                    }
                }
            }
        }
    }
}
```



```

        for(int q=0;q<rozmiarMaski;q++){
            for(int w=0;w<rozmiarMaski;w++){
                waga+=a[q][w]*maska[q][w];
            }
        }
        for(int x:czerwoki){
            if(waga==x)
                outtmp.setRGB(i, j, czerwony);
        }
    }
}

out=RGB.powiekszKopiujac(outtmp,0);
for(int color = 0;color<3;color++){
    //Ustawianie kolorow: 0-zielone 1-niebieskie 2-czerwone
    r=0;g=0;b=0;
    if(color==0){r=255;g=0;b=0;}
    if(color==1){r=0;g=255;b=0;}
    if(color==2){r=0;g=0;b=255;}

    for (int i = 1; i < width-1; i++) {
        for (int j = 1; j < height-1; j++) {
            // Wazystkie kolorowe piksele
            if (RGB.getR(out.getRGB(i, j))==r && RGB.getG(out.getRGB(i, j))==g && RGB.getB(out.getRGB(i, j))==b){
                waga = 0;
                for(int q=0;q<rozmiarMaski;q++){
                    for(int w=0;w<rozmiarMaski;w++){
                        a[q][w] = RGB.binar(RGB.getR(outbin.getRGB(i-rozmiarMaski/2+q,j-rozmiarMaski/2+w)));
                    }
                }

                for(int q=0;q<rozmiarMaski;q++){
                    for(int w=0;w<rozmiarMaski;w++){
                        waga+=a[q][w]*maska[q][w];
                    }
                }
                for(int x:wyciecia){
                    if(waga==x){
                        outtmp.setRGB(i, j,bialy);
                        outbin.setRGB(i, j,bialy);

                        straznik++;
                        zmiany=true;
                    }
                }
            }
        }
    }

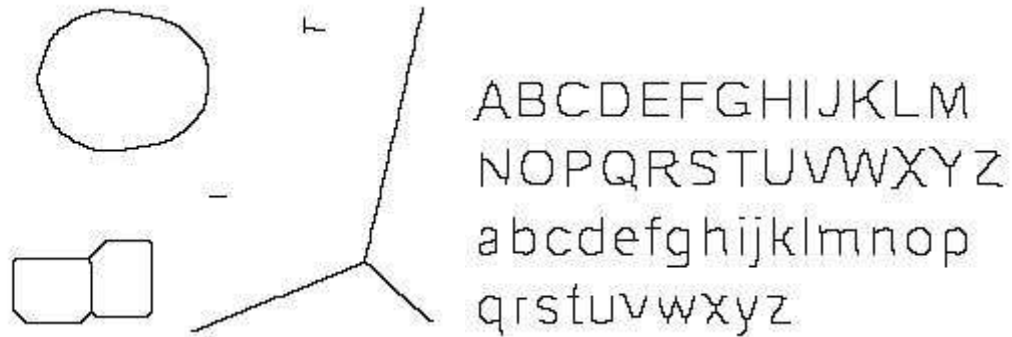
}

}while(zmiany==true);
long end_time = System.currentTimeMillis();
System.out.println("Wykonanie w : "+(end_time-start_time)+"ms");

return outbin;
}
}

```

## Algorytm K3M



Kod programu :

```
public static BufferedImage K3M ( BufferedImage in){  
  
    BufferedImage out=new BufferedImage( in.getWidth(), in.getHeight(), in.getType());  
    BufferedImage out1;//RGB.powiekszSialymi(in, 1);  
    out = RGB.powiekszKopiujac(in,0);  
    out1 = RGB.powiekszKopiujac(out,0);  
    int width;  
    int height;  
    width=in.getWidth();  
    height=in.getHeight();  
    int waga=0;  
  
    boolean zmiany;  
    do {  
        zmiany = false;  
        int r,g,b;  
        int straznik = 0;  
        //szukanie brzegowych pikseli i zamienianie ich na czerwone  
  
        for (int i = 3; i < width - 3; i++) {  
            for (int j = 3; j < height - 3; j++) {  
                //  
  
                if (RGB.getB(out.getRGB(i, j))==0){  
  
                    int a [][] = { {RGB.getB(out.getRGB(i-1, j-1))/255, RGB.getB(out.getRGB(i, j-1))/255, RGB.getB(out.getRGB(i+1, j-1))/255},  
                                   {RGB.getB(out.getRGB(i-1, j))/255, RGB.getB(out.getRGB(i, j))/255, RGB.getB(out.getRGB(i+1, j))/255},  
                                   {RGB.getB(out.getRGB(i-1, j+1))/255, RGB.getB(out.getRGB(i, j+1))/255, RGB.getB(out.getRGB(i+1, j+1))/255}  
                                };  
  
                    waga=0;  
                    for(int k=0;k<3;k++){  
                        for(int l=0;l<3;l++){  
                            waga+=maska[k][l]*Math.abs(a[k][l]-1);  
                        }  
                    }  
  
                    for(int x:A[0]){  
                        if (waga==x){  

```

```

        out.setRGB(i,j,new Color(255,0,0).getRGB());
    }
}

//
// pierwszy krok w celu wycięcia pikseli
for(int licz=1;licz<6;licz++) {
    for (int i=3; i< width-3; i++){
        for ( int j=3; j< height-3; j++){
            // tu uzupełnij kod
            waga=0;

            if((RGB.getR(out.getRGB(i, j)))==255 && (RGB.getB(out.getRGB(i, j)))==0){

                int a [][] = {
                    {RGB.getB(out.getRGB(i-1, j-1))/255, RGB.getB(out.getRGB(i, j-1))/255, RGB.getB(out.getRGB(i+1, j-1))/255},
                    {RGB.getB(out.getRGB(i-1, j))/255, RGB.getB(out.getRGB(i, j))/255, RGB.getB(out.getRGB(i+1, j))/255},
                    {RGB.getB(out.getRGB(i-1, j+1))/255, RGB.getB(out.getRGB(i, j+1))/255, RGB.getB(out.getRGB(i+1, j+1))/255}
                };

                for(int k=0;k<3;k++){
                    for(int l=0;l<3;l++){
                        waga+=maska[k][l]*Math.abs(a[k][l]-1);
                    }
                }

                for(int x:A[licz]){
                    if(waga==x){
                        out.setRGB(i,j,new Color(255,255,255).getRGB());
                        zmiany = true;
                    }
                }
            }
        }
    }
}

```

```

//zamiana pozostałych czerwonych pikseli na czarne
for (int i=1; i<width-2 ;i++){
    for ( int j=1; j< height-2; j++){
        //tu uzupełnij kod
        // wartosc red = 255 ale pozostale piksele musza byc 0 , dla białego wszystkie 255 wiec red rowniez
        // to ze red = 255 niewystarczające bo wszystkie białe rowniez beda sie kwalifikowaly
        if(RGB.getR(out.getRGB(i, j))==255 && RGB.getG(out.getRGB(i, j))==0){
            out.setRGB(i,j,new Color(0,0,0).getRGB());
        }
    }
}

straznik++;
if(straznik%10==0){
    try{
        ImageIO.write(out,"bmp", new File(straznik+"out.bmp"));
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}

//wszystkie czarne piksele, których waga znajduje sie w masce Alpix
//sa usuwane dopoki zachodza zmiany powtarzamy
while (zmiany==true);

do {
    zmiany=false;
    for ( int i=3; i<width-3; i++){
        for ( int j=3; j<height-3; j++){
            // tu uzupełnij kod

            if(RGB.getB(out.getRGB(i, j))==0){
                waga = 0;
                int a [][] = {
                    {RGB.getB(out.getRGB(i-1, j-1))/255, RGB.getB(out.getRGB(i, j-1))/255, RGB.getB(out.getRGB(i+1, j-1))/255},
                    {RGB.getB(out.getRGB(i-1, j))/255, RGB.getB(out.getRGB(i, j))/255, RGB.getB(out.getRGB(i+1, j))/255},
                    {RGB.getB(out.getRGB(i-1, j+1))/255, RGB.getB(out.getRGB(i, j+1))/255, RGB.getB(out.getRGB(i+1, j+1))/255}
                };

                for(int k=0;k<3;k++){
                    for(int l=0;l<3;l++){

```

```

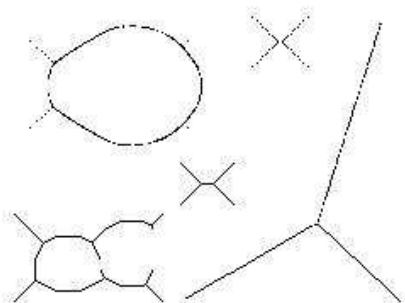
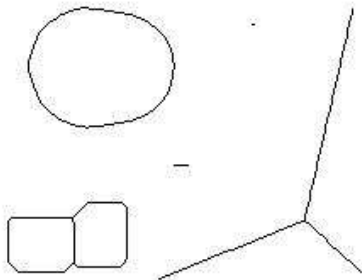
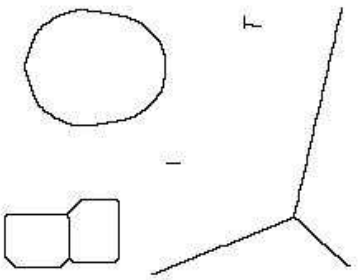
        waga += maska[k][1] * Math.abs(a[k][1] - 1);
    }

    for (int x: Alpix) {
        if (waga == x) {
            out.setRGB(i, j, new Color(255, 255, 255).getRGB());
            zmiany = true;
        }
    }
}

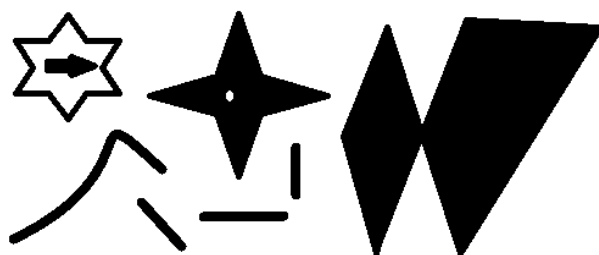
while (zmiany == true);
return out;
}

```

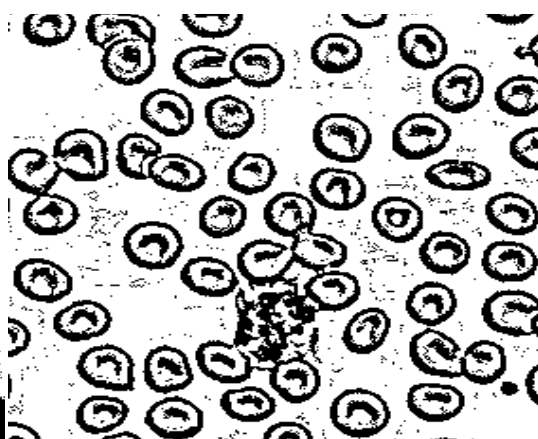
## Porównanie metod :

Przez maskę		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
KMM		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
K3M		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z

Obrazy do analizy o różnych własnościach :

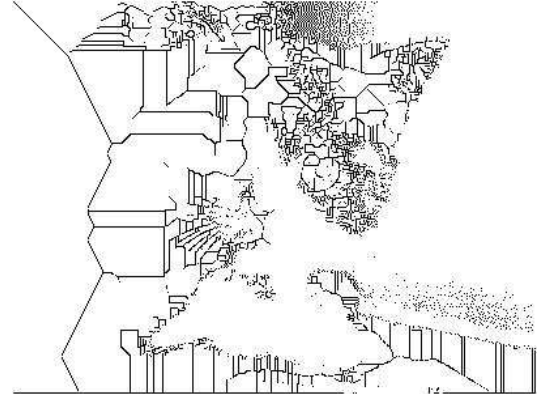
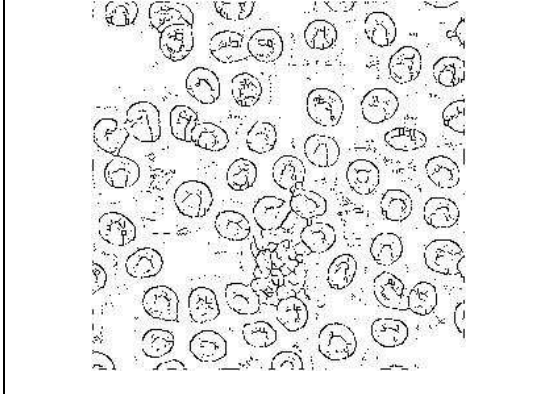
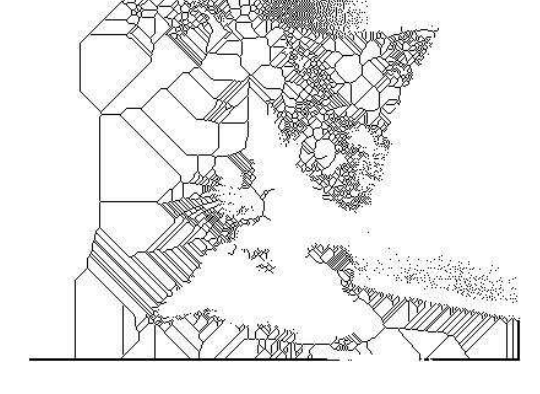
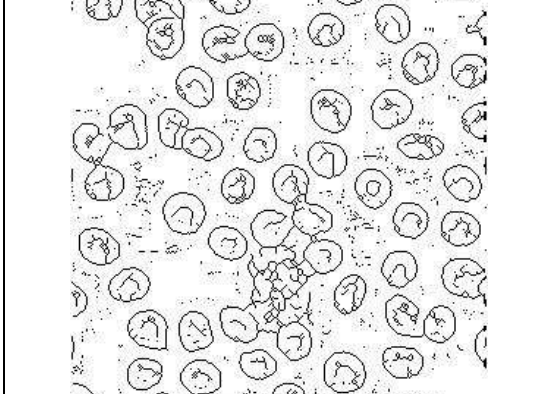
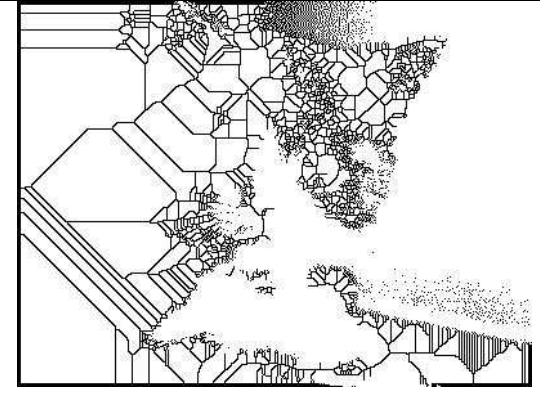
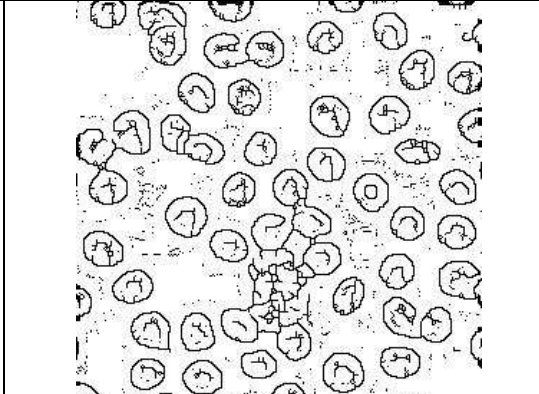


Ala ma kota, kot ma Alę!  
**Ala ma kota, kot ma Alę!**  
 Ala ma kota, kot ma Alę!  
*Ala ma kota, kot*  
*ma Alę!*



Przez maskę		Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! <i>Ala ma kota, kot</i> <i>ma Alę!</i>
KMM		Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! <i>Ala ma kota, kot</i> <i>ma Alę!</i>
K3M		Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! Ala ma kota, kot ma Alę! <i>Ala ma kota, kot</i> <i>ma Alę!</i>



Przez maskę		
KMM		
K3M		

## Wnioski (szkieletyzacja liter)

Algorytmy szkieletyzacji co do skuteczności plasują się następująco :

Dla skomplikowanych obiektów wszystkie algorytmy radzą sobie dość dobrze, możemy zauważyć że szkieletyzacja przez maskę dokonuje analizy obrazu bardziej w płaszczyźnie OX OY , natomiast algorytm KMM działa w płaszczyźnie skośnej, algorytm K3M łączy te dwie metody co widać na obrazie "Kota" przy porównaniu metod w tabeli powyżej.

Co do szkieletyzacji tekstów\liter :

przez maskę - wypada najslabiej, gubi znaczą część informacji.

KMM - wypada dosyć dobrze ale gubi informacje które mogą być niezbędne przy gorszej jakości tekstu.

K3M - zachowuje najwięcej informacji szkieletyzacja.

Zaletą szkieletyzacji przez maskę jest z pewnością szybkość działania, dla większych obrazów porównanie zarówno z KMM, jak również z K3M daje sporą różnicę. KMM działa nieco szybciej od K3M, ale ma tendencję do gubienia informacji. Dla poprawy jakości algorytmów możemy dokonać pewnych zmian w maskach likwidacji nadmiarowych sąsiadów (A1pix) - z pewnością może to wpłynąć na końcowy efekt szkieletyzacji.

## Ad II. Segmentacja

Analizowane obrazy :

<b>ABCDEFGHIJKLM</b>	<b>Ala ma kota, kot ma Alę!</b>
<b>NOPQRSTUVWXYZ</b>	<b>Ala ma kota, kot ma Alę!</b>
<b>abcdefghijklmnp</b>	<b>Ala ma kota, kot ma Alę!</b>
<b>qrstuvwxyz</b>	<i>Ala ma kota, kot ma Alę!</i>

Pierwszym etapem było wydzielenie wierszy z analizowanego tekstu :

<b>ABCDEFGHIJKLM</b>	<b>NOPQRSTUVWXYZ</b>
<b>abcdefghijklmnp</b>	<b>qrstuvwxyz</b>

Kod programu realizujący zadanie :

```
public static List<BufferedImage> wydobycieWierszy(BufferedImage in) {  
  
    BufferedImage outtmp = new BufferedImage(in.getWidth(), in.getHeight(), in.getType());  
    BufferedImage out = new BufferedImage(in.getWidth(), in.getHeight(), in.getType());  
    outtmp = RGB.powiekszKopiujac(in, 0);  
    out = RGB.powiekszKopiujac(in, 0);  
  
    int width;  
    int heighth;  
    width = in.getWidth();  
    heighth = in.getHeight();  
  
    int waga = 0;  
  
    int r, g, b;  
    int straznik = 0;  
  
    //Szukanie po wierszach i kopiowanie do nowego obrazka  
    /**  
     * Jesli posiada czarny piksel tzn ze jest litera i do listy dodajemy  
     * wartosc tego wiersza pozostale wiersze wypelniamy na zolto  
     */  
    HashMap<Integer, ArrayList<Integer>> czarneWiersze = new HashMap<>();  
    List<Integer> czarne = new ArrayList<>();  
    List<BufferedImage> wierszeObrazy = new ArrayList<>();  
    //wiersze ktore posiadaja czarne piksele  
    for (int i = 0; i < width; i++) {  
        for (int j = 0; j < heighth; j++) {  
            if (RGB.getR(outtmp.getRGB(i, j)) == 0) {  
                if (!czarne.contains(j)) {  
                    czarne.add(j);  
                }  
            }  
        }  
    }  
}
```



```

//wypelnianie pozostalych wierszy
for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
        if (!czarne.contains(j)) {
            out.setRGB(i, j, zolty);
        }
    }
}

Collections.sort(czarne);

int first, current, previous;
int licznikwierszy = 1;
first = czarne.get(0);
previous = czarne.get(0);

for (int i = 0; i < czarne.size(); i++) {
    //System.out.println(czarne.get(i));
    current = czarne.get(i);

    if (current - previous > 1 || i == czarne.size() - 1) {
        //Jesli koniec petli ostatni wiersz obrazka
        if (i == czarne.size() - 1) {
            previous = current;
        }

        BufferedImage temp = new BufferedImage(out.getWidth(), (previous - first + 1), out.getType());
        for (int k = 0; k < width; k++) {
            for (int l = first, m = 0; l <= previous; l++) {
                temp.setRGB(k, m++, out.getRGB(k, l));
            }
        }

        wierszeObrazy.add(temp);
        first = current;
    }
}

```

```

    }
    previous = current;
}

try {
    for (int i = 0; i < wierszeObrazy.size(); i++) {
        ImageIO.write(wierszeObrazy.get(i), "jpg", new File("out/wiersz" + (i + 1) + ".jpg"));
    }
} catch (Exception e) {
    e.getMessage();
}

return wierszeObrazy;
}

```

Następnie z wydobytych wierszy należy wydobyć litery :

Przerwy między nimi miały być oznaczone na żółto natomiast obszary ponad i poniżej oznaczone na czerwono :

Wynik działania programu :

**abcdefghijklmnop**

Kod programu :

```

public static BufferedImage wydobycieLiter(BufferedImage in) {

    BufferedImage outtmp = new BufferedImage(in.getWidth(), in.getHeight(), in.getType());
    BufferedImage out = new BufferedImage(in.getWidth(), in.getHeight(), in.getType());
    outtmp = RGB.powiekszKopiujac(in, 0);
    out = RGB.powiekszKopiujac(in, 0);

    int width;
    int heighth;
    width = in.getWidth();
    heighth = in.getHeight();

    int waga = 0;

    int x, g, b;
    int straznik = 0;

    //Szukanie po wierszach i kopiowanie do nowego obrazka
    /**
     * Jesli posiada czarny piksel tzn ze jest litera i do listy dodajemy
     * wartosc tego wiersza pozostale wiersze wypelniamy na zolto
     */
    HashMap<Integer, ArrayList<Integer>> czarneWiersze = new HashMap<>();
    List<Integer> czarne = new ArrayList<>();
    List<List<Integer>> odcinki = new ArrayList<>();
    //wiersze ktore posiadaja czarne piksele
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < heighth; j++) {
            if (RGB.getR(outtmp.getRGB(i, j)) == 0) {
                if (!czarne.contains(i)) {
                    czarne.add(i);
                }
            }
        }
    }
}

```

```

//wypelnianie pozostalych wierszy
for (int i = 0; i < width; i++) {
    for (int j = 0; j < heighth; j++) {
        if (!czarne.contains(i)) {
            out.setRGB(i, j, zolty);
        }
    }
}

Collections.sort(czarne);
//znacznik konca listy dodana wartosc wieksza o 2 od ostatniego wyrazu
//zeby poprawnie przeszla petla i zauwazyla roznice w celu zaznaczenia kolumny
czarne.add(czarne.get(czarne.size() - 1) + 2);

int first, previous, current, count, max, roznica;
first = czarne.get(0);
previous = czarne.get(0);
max = heighth;
count = 0;

for (int k : czarne) {
    current = k;
    roznica = current - previous;
    if (roznica > 1) {
        List<Integer> tempList = new ArrayList<>();
        for (int i = first; i <= previous; i++) {
            tempList.add(i);
        }
        odcinki.add(tempList);
        first = current;
    }
    previous = current;
}

//sprawdzanie kazdego odcinka
boolean zawieraCzarne = false;

```

```

//sprawdzanie każdego odcinka
boolean zawieraCzarne = false;
for (List l : odcinki) {
    for (int j = 0; j < heighth; j++) {
        zawieraCzarne = false;

        for (int i = 0; i < l.size(); i++) {
            int w = (int) l.get(i);
            if (RGB.getB(out.getRGB(w, j)) == 0) {
                zawieraCzarne = true;
            }
        }

        if (zawieraCzarne == false) {
            for (int i = 0; i < l.size(); i++) {
                int w = (int) l.get(i);
                out.setRGB(w, j, czerwony);
            }
        }
    }
}

return out;
}

```

W mojej wersji programu wynikowe wiersze z wydzielonymi literami do analizy łączyłem w jeden wynikowy obraz :



Kod programu :

```

public static BufferedImage joiner(List<BufferedImage> in) {
    BufferedImage out;
    int width = 0;
    int height = 0;
    int startHeight = 0;
    int endHeight = 0;

    for (BufferedImage im : in) {
        width = im.getWidth();
        height += im.getHeight();
    }

    out = new BufferedImage(width, height, in.get(0).getType());

    startHeight = 0;
    endHeight = 0;

    for (BufferedImage im : in) {
        endHeight += im.getHeight();

        for (int i = 0; i < width; i++) {
            for (int j = startHeight, k = 0; j < endHeight; j++, k++) {
                out.setRGB(i, j, im.getRGB(i, k));
            }
        }

        startHeight += im.getHeight();
    }

    return out;
}

```

## Kod głównej funkcji zarządzającej wydzielaniem :

```
public static BufferedImage szkieletyzacja(BufferedImage in) {  
  
    List<BufferedImage> listaWierszy = new ArrayList<>();  
    List<BufferedImage> listaLiter = new ArrayList<>();  
    HashMap<String, ArrayList<Integer>> czarneWiersze = new HashMap<>();  
    BufferedImage out;  
  
    listaWierszy = wydobycieWierszy(in);  
  
    for (BufferedImage l : listaWierszy) {  
        listaLiter.add(wydobycieLiter(l));  
    }  
  
    out = joiner(listaLiter);  
    return out;  
}
```