

Aplikace načítá rastrový obrázek a detekuje jednotlivá písmena anglické abecedy. Program je napsán v programovacím Jazyce JAVA. Autor zde demonstruje své schopnosti algoritmizace.

Rozpoznávání písmen z png

Dokumentace k algoritmu

Jonáš Bartoň

Obsah

1.	Aplikace na detekci písmen.....	2
1.1	Co aplikace umožňuje.....	2
1.2	Princip algoritmu.....	3
1.2.1	Detekce řádků.....	4
1.2.2	Detekce sloupců (písmen).....	4
1.2.3	Detekce písmene s porovnáním z předlohy.....	5
1.2.4	Algoritmus rozpoznání písmene.....	6
2.	Aplikace na mazání obrazců.....	8
2.1	Účel aplikace.....	8
2.2	Popis algoritmu, data.	9
2.2.1	Data obrázku – všechny pixely.....	9
2.2.2	Data obrázku – pouze potřebné pixely.....	9
2.2.3	Rozdělení dat podle barev.....	10
2.2.4	Optimalizace algoritmu.....	11

1. Aplikace na detekci písmen.

1.1 Co aplikace umožňuje.

Aplikace umožňuje detekovat písmena z rastrového obrázku do textové podoby. Zdrojový kód je v programovacím jazyce JAVA. Aplikace je uložena zde:

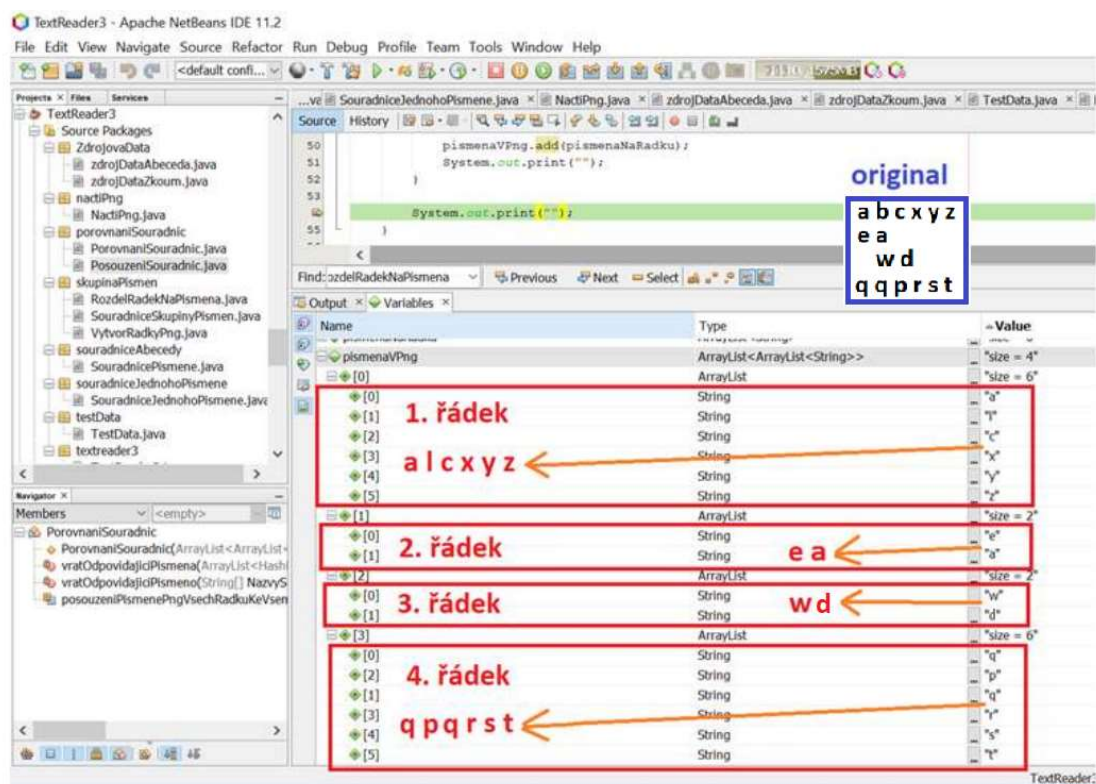
<https://github.com/jonasRower/Read-png-JAVA.git>

Zdrojový testovaný obrázek je zde:

a b c x y z
e a
w d
q q p r s t

Obr. 1.1 – Obrázek (data) určený k testování aplikace.

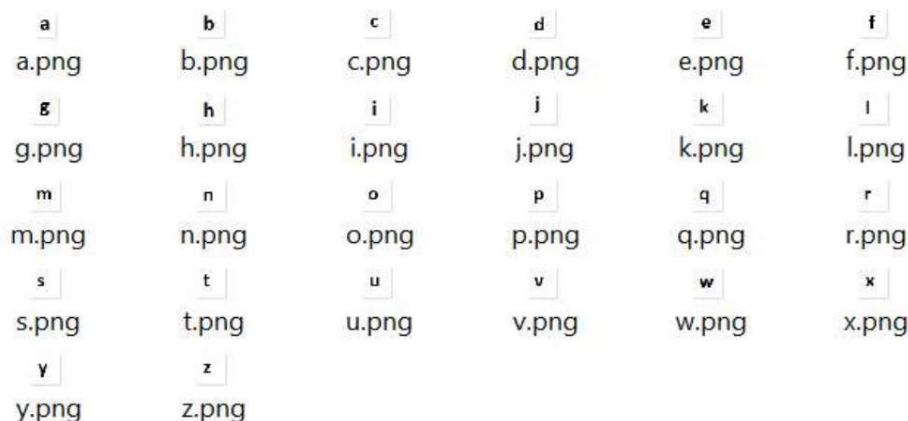
Výstup zobrazuji v debug-módu:



Obr. 1.2 – Rozpoznání obrázku – výstup z debug módu.

1.2 Princip algoritmu.

Algoritmus je založen pouze na principu porovnávání písmen mezi sebou a to mezi předlohou a testovaným obrázkem. Předlohou se tedy rozumí složka s obrázky abecedy, kde každé písmeno je uloženo jako jeden soubor png.

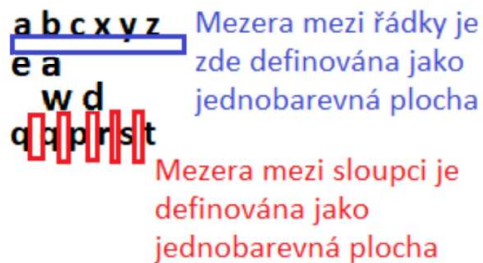


Obr. 1.3 – Sada vzorových obrázků (písmen), na základě jejich srovnání jsou detekována jednotlivá písmena.

Z této úvahy také samozřejmě vychází ten fakt, že algoritmus je značně omezen pouze na daný typ fontu a výšku písma. Dokonalejší algoritmus je již navržen, avšak nenaprogramován a není součástí této dokumentace.

Nicméně pro porovnání písmen, je potřeba testovaný obrázek rozložit na jednotlivé řádky a na každém řádku pak na jednotlivé sloupce (tedy písmena). Jednotlivé řádky jsou zde definovány způsobem, že mezera mezi řádky je pouze jednou barvou (což je opět velice zjednodušené řešení, platné pro úzkou skupinu možných případů - rozšíření tohoto algoritmu je popsáno v kapitole 2.)

Princip rozložení, je zobrazen na následujícím obrázku 1.4.



Obr. 1.4 – Rozložení obrázku na jednotlivá písmena

1.2.1 Detekce řádků.

Testovaný obrázek se rozloží na jednotlivé řádky a na řádcích na jednotlivé sloupce. Vyberou se všechny pixely jiné než bílé barvy (opět je zde limitující zjednodušení, co se týče barvy) a ty se porovnají s jednotlivými obrázky z předlohy (abecedy).

Na obrázku níže jsou ukázány data y-souřadnice pixelů vždy na hoře a na spodě řádku. Data jsou uchována v třídě VytvorRadkyPng.java.

The screenshot shows the IDE interface with the following data:

YprvniAPosledniPixelRadkuPole	YprvniAPosledniPixelRadkuPole	YprvniAPosledniPixelRadkuPole
[0]	HashMap	"size = 4"
[0]	HashMap\$Node	"YzacatkuRadku => 5"
[1]	HashMap\$Node	"YkonecRadku => 25"
[1]	HashMap	"size = 2"
[0]	HashMap\$Node	"YzacatkuRadku => 33"
[1]	HashMap\$Node	"YkonecRadku => 44"
[2]	HashMap	"size = 2"
[0]	HashMap\$Node	"YzacatkuRadku => 55"
[1]	HashMap\$Node	"YkonecRadku => 66"
[3]	HashMap	"size = 2"
[0]	HashMap\$Node	"YzacatkuRadku => 75"
[1]	HashMap\$Node	"YkonecRadku => 93"

Diagram illustrating row detection with letters and their corresponding y-coordinates:

- Row 1: a b c x y z (y=5 to 25)
- Row 2: e a (y=33 to 44)
- Row 3: w d (y=55 to 66)
- Row 4: q q p r s t (y=75 to 93)

Obr. 1.5 – ArrayList YprvniAPosledniPixelRadkuPole obsahuje indexy pixelů y-ové souřadnice začátku a konce řádku v obrázku.

1.2.2 Detekce sloupců (písmen).

Obdobně se jedná i o data na rozmezí sloupců. Obrázek níže ukazuje indexy x-souřadnice pixelů vlevo a vpravo každého sloupce daného řádku, zde se jedná o řádek první (obsahující 6 písmen). Data jsou uchována v třídě RozdelRadekNaPismena.java. Samozřejmě pro každý řádek se zde volá nová instance třídy.

The screenshot shows the IDE interface with the following data:

XprvniAPosledniPixelSloupcePole	XprvniAPosledniPixelSloupcePole	XprvniAPosledniPixelSloupcePole
[0]	HashMap	"size = 6"
[0]	HashMap\$Node	"size = 2"
[1]	HashMap\$Node	"XkonecSloupce => 13"
[1]	HashMap	"XzacatkuSloupce => 2"
[1]	HashMap	"size = 2"
[0]	HashMap\$Node	"XkonecSloupce => 32"
[1]	HashMap\$Node	"XzacatkuSloupce => 20"
[2]	HashMap	"size = 2"
[0]	HashMap\$Node	"XkonecSloupce => 47"
[1]	HashMap\$Node	"XzacatkuSloupce => 37"
[3]	HashMap	"size = 2"
[0]	HashMap\$Node	"XkonecSloupce => 63"
[1]	HashMap\$Node	"XzacatkuSloupce => 52"
[4]	HashMap	"size = 2"
[0]	HashMap\$Node	"XkonecSloupce => 80"
[1]	HashMap\$Node	"XzacatkuSloupce => 67"
[5]	HashMap	"size = 2"

Obr. 1.6 – ArrayList XprvniAPosledniPixelSloupcePole obsahuje indexy pixelů x-ové souřadnice začátku a konce sloupce v daném řádku.

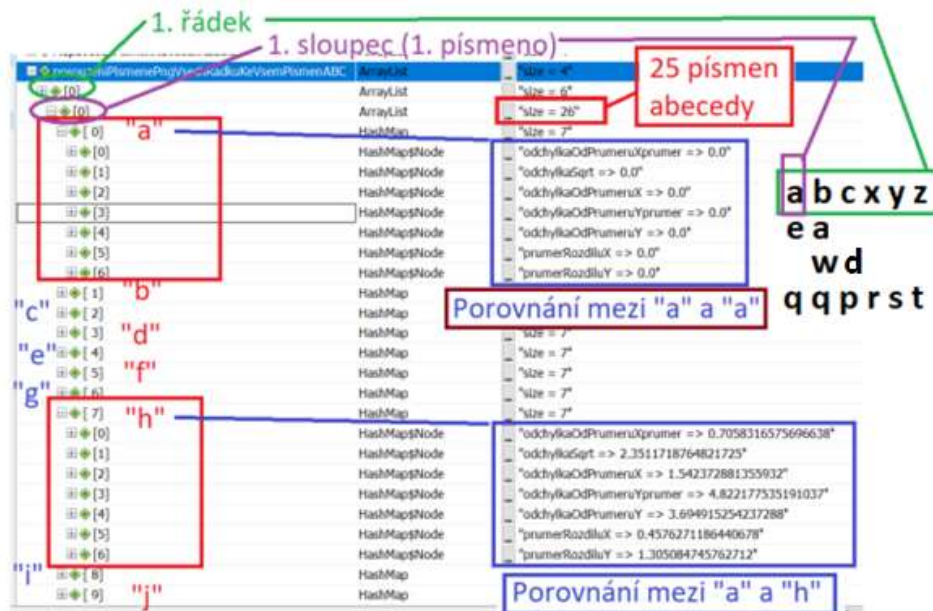
1.2.3 Detekce písmene s porovnáním z předlohy.

ArrayList posouzeniPismenePngVsechRadkuKeVsemPismenABC obsahuje data porovnání daného písmene testovaného obrázku s každým obrázkem z předlohy (abecedy). Jedná se tedy o ArrayList s 4-mi indexy (jednotlivé řádky) a na příslušném řádku je příslušný počet písmen – 26 tj. 26 písmen anglické abecedy. Data jsou uložena ve třídě PosouzeniSouradnic.java.



Obr. 1.7 – ArrayList posouzeniPismenePngVsechRadkuKeVsemPismenABC obsahující data testovaného obrázku

Obrázek níže (Obr. 1.8) již ukazuje rozbalenou strukturu dat, tedy konkrétně prvního písmene na prvním řádku, jedná se o písmeno "a". Všimneme si, že každá úroveň ArrayListu uchovávající data jednoho písmene, obsahuje přesně 26 indexů - to přesně odpovídá 26-ti písmenům anglické abecedy. Na obrázku níže jsme tedy rozbalili první písmeno na prvním řádku a to s indexem 0 a 7. Všimneme si, že data 0-tého indexu jsou rovny 0, zatímco na 7-mém indexu se od 0 liší. Je to proto, že 0-tý index zobrazuje porovnání mezi písmeny "a" a "a" a 7-mý index zobrazuje porovnání mezi písmeny "h" a "a" (čísle se od nuly, takže 0 = "a" a 7 = "h"). Jelikož však máme na testovacím obrázku první písmeno prvního řádku "a", pak všechny porovnávané faktory (mezi "a" a "a" => 0) zatímco v opačném případě rostou od nuly. Těmi porovnávanými faktory se rozumí odchylky od jednotlivých pixelů mezi písmenem z testovaného obrázku a mezi písmenem z předlohy (abecedy).



Obr. 1.8 – Detekování jednotlivých písmen zdrojového obrázku

1.2.4 Algoritmus rozpoznání písmene.

Samozřejmě nedá se s jistotou říci, že vždy bude vyhovující to písmeno s faktory rovny 0 ($= 0$), jelikož to nemusí být vždy splněno. Nemusí to být vždy splněno z toho důvodu, že obrázek v předloze může být jiného fontu a rozdíly v odchylkách jednotlivých pixelů nemusí tedy vést k nule, ale mohou být k nule blízké, nikoliv rovné. Avšak z pravděpodobnostního hlediska, je nepravděpodobné, aby nějaké jiné písmeno mělo menší odchylky v rozdílu souřadnic pixelů, než písmeno příslušející, ačkoliv s jiným fontem.

Za tohoto předpokladu tedy algoritmus funguje tím způsobem, že porovnává všechny faktory, v našem případě, mezi "a" a "h" a vybere to písmeno, které má faktory nejmenší (nikoliv rovny nule). Dále jen podotkněme, že porovnávání faktorů máme vícero a porovnáním se zabývá speciální algoritmus, který vybírá příslušná písmena.

Obrázek níže již ukazuje vybraná písmena v proměnné `pismenaVPng` (třída `porovnavaniSouradnic.java`). Je evidentní, že datová struktura zůstala zachována, je shodná jako s daty `posouzeniPismenePngVsechRadkuKeVsemPismenABC`, jen místo 26 indexů abecedy obsahuje již vybraná písmena, ta písmena, která byla vyhodnocena jako písmena s minimálními odchylkami oproti předloze.

TextReader3 - Apache NetBeans IDE 11.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Source Packages

- zdrojovaData
 - zdrojDataAbeceda.java
 - zdrojDataZkousm.java
- nactiPng
 - NactiPng.java
- porovnaviSouradnic
 - PorovnaviSouradnic.java
 - PosouzeniSouradnic.java
- skupinaPismen
 - RozdelRadekNaPismena.java
 - SouradniceSkupinyPismen.java
 - VytvorRadkyPng.java
- souradniceAbecedy
 - SouradnicePismene.java
 - SouradniceJednohoPismene.java
- testData
 - TestData.java
- textreader3
 - textreader3

Members

- PorovnaviSouradnic
 - PorovnaviSouradnic(ArrayList<ArrayList>)
 - vratiOdpovidajiciPismena(ArrayList<Hash
 - vratiOdpovidajiciPismena(String[] NazvyS
 - posouzeniPismenePngVsechRadkuKeVse

Source

```

50  pismenaVPng.add(pismenaNaRadku);
51  System.out.print("");
52
53
54
55  System.out.print("");
56
57
58
59
60

```

Find: zdrojRadekNaPismena Previous Next Select

Output Variables

Name	Type	Value
pismenaVPng	ArrayList<ArrayList<String>>	"size = 4"
[0]	ArrayList	"size = 6"
[0]	String	"a"
[1]	String	"i"
[2]	String	"c"
[3]	String	"x"
[4]	String	"y"
[5]	String	"z"
[1]	ArrayList	"size = 2"
[0]	String	"e"
[1]	String	"a"
[2]	ArrayList	"size = 2"
[0]	String	"w"
[1]	String	"d"
[3]	ArrayList	"size = 6"
[0]	String	"q"
[2]	String	"p"
[1]	String	"r"
[3]	String	"s"
[4]	String	"t"
[5]	String	" "

original

a b c x y z
e a
w d
q p r s t

1. řádek
a l c x y z

2. řádek
e a

3. řádek
w d

4. řádek
q p q r s t

TextReader3

Obr. 1.9 – Detekování jednotlivých písmen zdrojového obrázku

2. Aplikace na mazání obrázců.

2.1 Účel aplikace.

Právě vzhledem k nedokonalosti předchozího algoritmu, byl navržen další algoritmus za cílem odstranění jiných obrázců než písmen z obrázku. Jak jsme již zmiňovali v kapitole 1.1. , uvedli jsme, že mezera mezi řádky je definována jako plocha pouze jedné barvy. To pak ale znamená, že algoritmus níže nemůže detekovat rozhraní jednotlivých řádků, jelikož oblast mezi řádky je jiné barvy než jednobarevná.

Jedná se o tento testovací obrázek.



Zdrojový kód je umístěn zde:

https://drive.google.com/drive/folders/1UWOo9A03cAsjS1QWWDtxIW75HL_vz9Y7

Obrázek níže zobrazuje daný problém. Obrázec modré barvy je po celé výšce obrázku, přesahuje tedy všechny jednobarevné řádky. Za tohoto předpokladu, nemůže být řádek detekován, jelikož, pomocí předchozího algoritmu, se mezera mezi řádky detekuje jako jednobarevná plocha. Právě za tímto cílem byl navržen algoritmus, který právě odmazává jiné obrázky než písmena a usnadňuje tak detekci jednotlivých řádků.

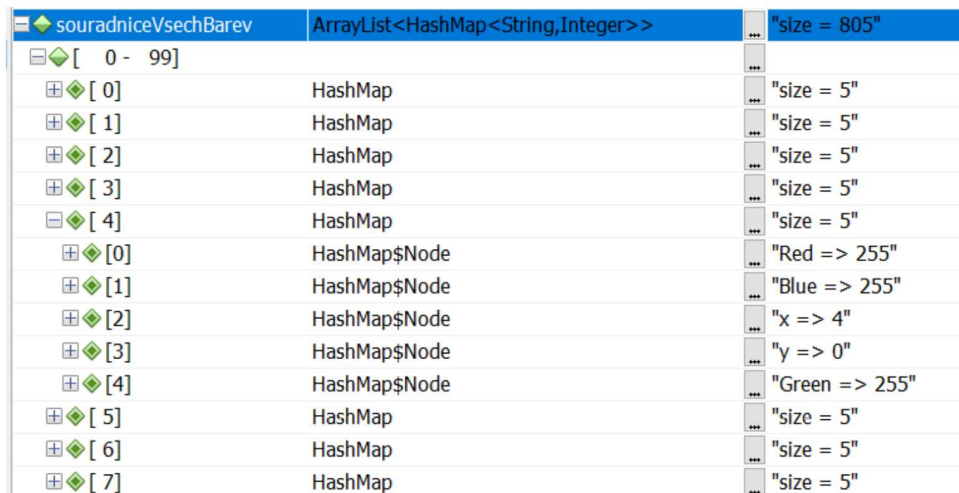


Obr. 2.1 – Mezery mezi řádky a sloupci nejsou jedné barvy, proto je třeba detekovat tyto obrázky a pixely jednotlivých obrázců přepisovat na barvu pozadí.

2.2 Popis algoritmu, data.

2.2.1 Data obrázku – všechny pixely.

Metoda Main, kde začíná program je ve třídě VektoryCar.java. Následně algoritmus ze zdrojového obrázku načítá všechna data (souřadnice a rgb všech pixelů) do ArrayListu souradniceVsechBarev.

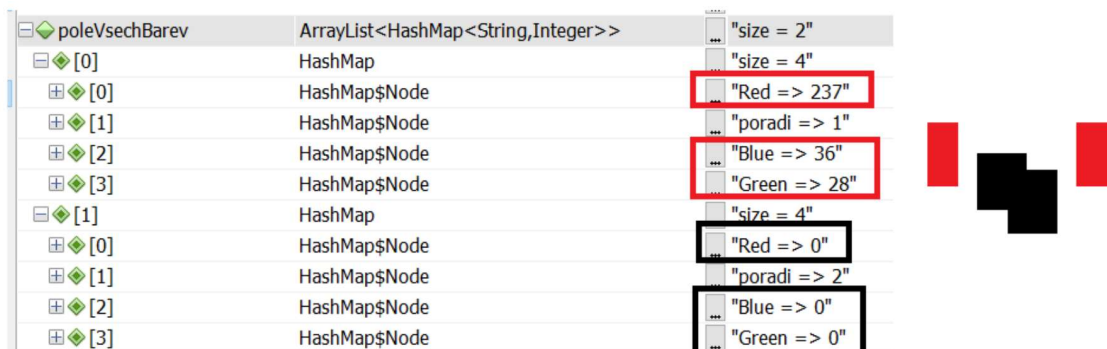


Index	Type	Value
[0 - 99]	ArrayList	size = 805
[0]	HashMap	size = 5
[1]	HashMap	size = 5
[2]	HashMap	size = 5
[3]	HashMap	size = 5
[4]	HashMap	size = 5
[4]	HashMap\$Node	Red => 255
[4]	HashMap\$Node	Blue => 255
[4]	HashMap\$Node	x => 4
[4]	HashMap\$Node	y => 0
[4]	HashMap\$Node	Green => 255
[5]	HashMap	size = 5
[6]	HashMap	size = 5
[7]	HashMap	size = 5

Obr. 2.2 – ArrayList souradniceVsechBarev obsahuje x-ovou a y-ovou souřadnici pixelu a hodnotu RGB.

2.2.2 Data obrázku – pouze potřebné pixely.

Následně se pracuje pouze s těmito daty a obrázek se již opětovně nenačítá. Získá se poleVsechBarev, tj. ArrayList uchovávající RGB všech barev. Algoritmus je stále i zde zjednodušený a uvažuje, že pozadí je vždy bílé, proto bílou barvu nezahrnuje. Obrázek níže zobrazuje RGB data všech barev v obrázku.



Index	Type	Value
[0]	ArrayList	size = 2
[0]	HashMap	size = 4
[0]	HashMap\$Node	Red => 237
[0]	HashMap\$Node	poradi => 1
[0]	HashMap\$Node	Blue => 36
[0]	HashMap\$Node	Green => 28
[1]	HashMap	size = 4
[1]	HashMap\$Node	Red => 0
[1]	HashMap\$Node	poradi => 2
[1]	HashMap\$Node	Blue => 0
[1]	HashMap\$Node	Green => 0

Obr. 2.3 – ArrayList poleVsechBarev obsahuje data jako ArrayList souradniceVsechBarev, pouze těch barev pixelů mimo barvu pozadí.

2.2.3 Rozdělení dat podle barev.

Data se třídí do ArrayListu `sousedilJinaBarvaPodleBarev`. Jednotlivé indexy odpovídají počtu indexů v `poleVsechBarev`. Poslední index je index barvy pozadí, který v `poleVsechBarev` není.

Index	Typ	Obsah
0	ArrayList	"size = 3"
1	ArrayList	"size = 68"
2	ArrayList	"size = 54"
3	ArrayList	"size = 122"

Index	Typ	Obsah
0	HashMap	"size = 2"
1	HashMap	"size = 4"
2	HashMap\$Node	"Red => 237"
3	HashMap\$Node	"poradi => 1"
4	HashMap\$Node	"Blue => 36"
5	HashMap\$Node	"Green => 28"
6	HashMap	"size = 4"
7	HashMap\$Node	"Red => 0"
8	HashMap\$Node	"poradi => 2"
9	HashMap\$Node	"Blue => 0"
10	HashMap\$Node	"Green => 0"

barva pozadí

Obr. 2.3 – ArrayList `sousedilJinaBarvaPodleBarev` seskupuje pixely jedné barvy do jednoho ArrayListu podřazené úrovně.

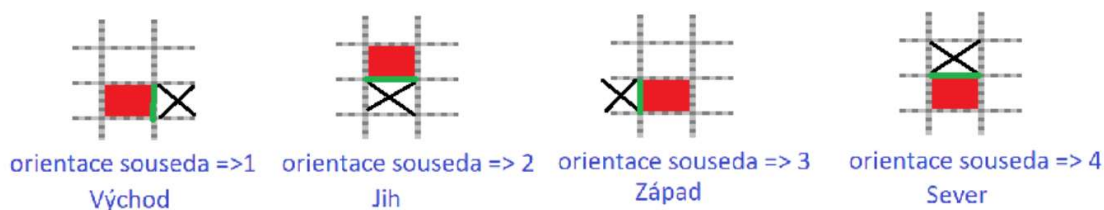
Tento ArrayList uchovává všechny pixely (HashMapy), které jsou na rozhraní dvou barev. Každá HashMapa tedy obsahuje vždy data dvojice pixelů, mezi nimiž je dané rozhraní. Význam jednotlivých Hash je uveden na obrázku níže:

Index	Typ	Obsah	Popis
0	ArrayList	"size = 3"	
1	ArrayList	"size = 68"	
2	ArrayList	"size = 13"	
3	HashMap	"y_Hlavni => 1"	y-souř. hlavního pixelu
4	HashMap\$Node	"jedna se o okraj plochy => 1"	x-souř. hlavního pixelu
5	HashMap\$Node	"x_Hlavni => 1"	y-souř. sousedního pixelu
6	HashMap\$Node	"y_Soused => 0"	
7	HashMap\$Node	"r_Soused => 255"	RGB hlavního pixelu (červená)
8	HashMap\$Node	"g_Soused => 36"	
9	HashMap\$Node	"b_Soused => 255"	RGB sousedního pixelu (bílá)
10	HashMap\$Node	"r_Soused => 255"	
11	HashMap\$Node	"orientace souseda => 2"	
12	HashMap\$Node	"x_Soused => 1"	x-souř. sousedního pixelu
13	HashMap\$Node	"y_Hlavni => 237"	
14	HashMap\$Node	"jedna se o okraj => 0"	

orientace souseda

Obr. 2.5 – Podúroveň ArrayListu `sousedilJinaBarvaPodleBarev` obsahuje x,y souřadnice a RGB aktuálního (hlavního) pixelu a pixelu vedlejšího.

U položky orientace souseda, jednotlivé strany jsou očíslovány čísly 1-4 ve směru hodinových ručiček od východu na sever, viz obrázek níže.



Obr. 2.6 – Sousední pixely jiné barvy než barvy pozadí, na hranici obrazce, jsou označeny podle jejich orientace k přilehlému okraji.

Následně se data třídí do ArrayListu `sousediStejnaBarvaPodleBarev`. Oproti `sousediJinaBarvaPodleBarev` se jedná o pixely, které mají ve všech 4 stranách kolem daného pixelu souseda stejné barvy, jedná se tedy o pixel, jak je zobrazeno na následujícím obrázku:



Obr. 2.7 – Sousední pixely jiné barvy než barvy pozadí, uvnitř obrazce. Zde není třeba rozlišovat orientaci.

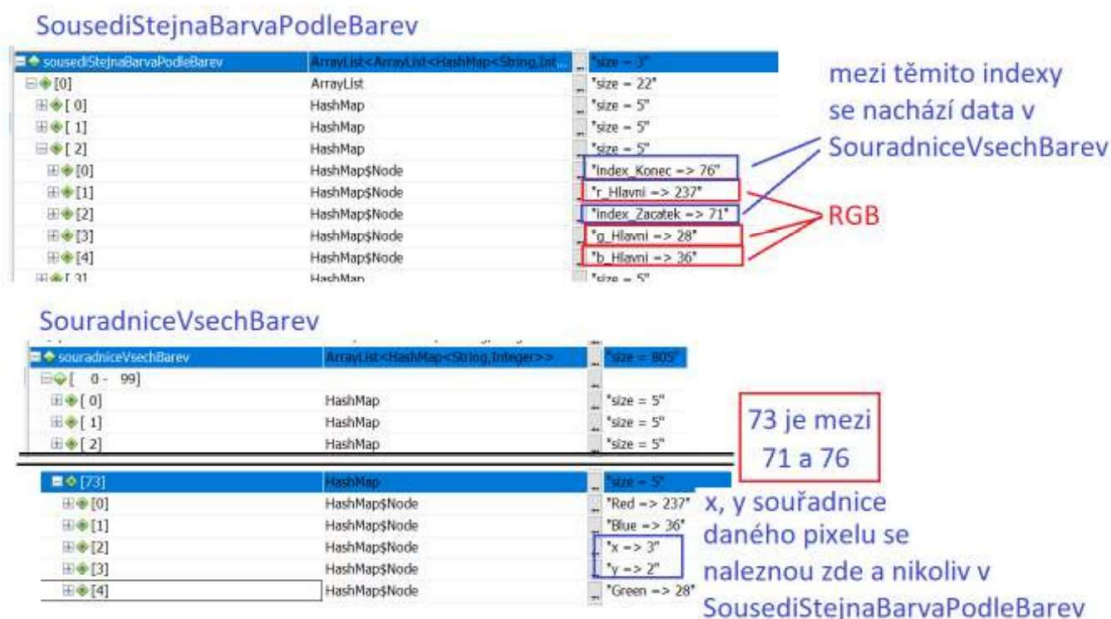
Stejně jako u `sousediJinaBarvaPodleBarev` jsou data roztržena podle barev.

<code>sousediStejnaBarvaPodleBarev</code>	<code>ArrayList<ArrayList<HashMap<String,Int...</code>	<code>"size = 3"</code>
<code>[0]</code>	<code>ArrayList</code>	<code>"size = 22"</code>
<code>[1]</code>	<code>ArrayList</code>	<code>"size = 14"</code>
<code>[2]</code>	<code>ArrayList</code>	<code>"size = 36"</code>

Obr. 2.8 – V ArrayListu `sousediStejnaBarvaPodleBarev` jsou uchovávány RGB a souřadnice těch pixelů, které spolu sousedí a jsou stejné barvy.

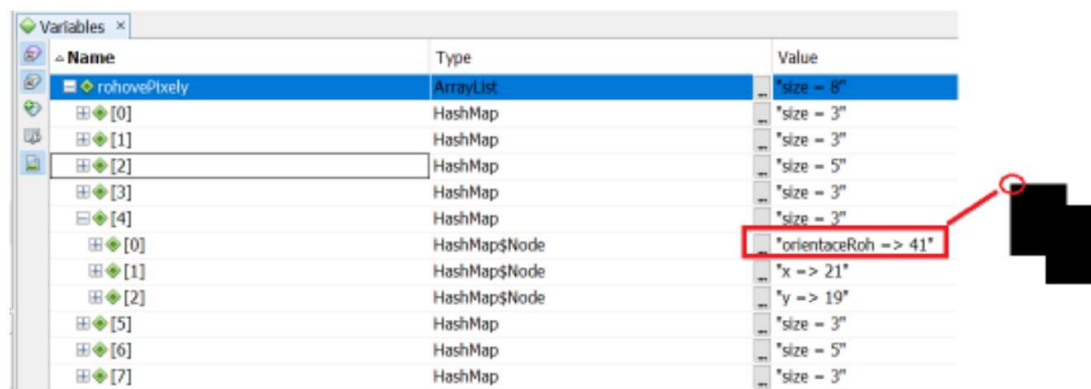
2.2.4 Optimalizace algoritmu.

Z důvodu rychlejšího algoritmu, jelikož se jedná o poměrně jednoduchá data, avšak častokrát násobně větší množství, než `sousediJinaBarvaPodleBarev`, hashmap uchovává pouze odkazy na intervaly mezi kterými indexy v ArrayListu `souradniceVsechBarev` se daná barva nachází, viz obrázek:



Obr. 2.9 – SousediStejnaBarvaPodleBarev uchovává informaci krajních indexů pixelů, mezi kterými se hledaný pixel nachází. Souřadnice daného pixelu je nutno dohledávat v ArrayListu SouradniceVsechBarev.

Dále ArrayList rohovePixely budeme také považovat za důležitá data. Tento ArrayList je umístěn v třídě OkrajKolemDokola, jehož instance se volá s každým obrazcem (v našem případě 3x - 2x pro červený obrazec a 1x pro černý). Právě pro černý obrazec (který je nejsložitější) máme data:



Obr. 2.9 – ArrayList rohovePixely uchovává informaci o x,y souřadnicích a orientaci rohového pixelu

Principiálně algoritmus funguje tím způsobem, že detekuje pixely v oblasti mezi rohovými pixely a přepisuje pole souradniceVsechBarev detekovaných pixelů na barvu pozadí, čímž pixely odmazává.