LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

TEKSTA ZĪMJU ATTĒLU UZLABOŠANA

KVALIFIKĀCIJAS DARBS

Autors: **Jevgenijs Jonass**
Stud. apl. nr.: jj07020
Darba vadītājs: Dr. mat. Paulis Ķikusts

RĪGA 2009

ANOTĀCIJA

Šī kvalifikācijas darba mērķis bija izstrādāt attēlu apstrādes programmas moduli, kas atbild par teksta zīmju attēlu struktūras analīzi, kā arī realizēt attēlu priekšapstrādes funkcijas, kas ietilpst šīs programmas galvenā moduļa sastāvā. Programmas izstrāde veikta prakses ietvaros Latvijas Universitātes Matemātikas un informātikas institūtā. Programma ir rakstīta programmēšanas valodā *Java*, kas ir augsta līmeņa objektorientēta programmēšanas valoda.

Atslēgvārdi: attēlu apstrāde, grafi, teksta atpazīšana

ABSTRACT

Objective of this qualification work was to develop a module for text character structure analysis, which is part of an image processing programme, and to develop functions of the main programme module which are responsible for image preprocessing. The programme was developed during practice in The Institute of Mathematics and Computer Sciences, University of Latvia (IMCS UL). The programme is written in *Java*, which is a high-level object-oriented programming language.

Keywords: image processing, graphs, text recognition

Satura rādītājs

Ievads

Šī kvalifikācijas darba mērķis bija izstrādāt attēlu apstrādes programmas moduli, kas atbild par teksta zīmju attēlu struktūras analīzi, kā arī realizēt attēlu priekšapstrādes funkcijas, kas ietilpst šīs programmas galvenā moduļa sastāvā. Programmas izstrāde veikta prakses ietvaros Latvijas Universitātes Matemātikas un informātikas institūtā. Programma ir rakstīta programmēšanas valodā *Java*, kas ir augsta līmeņa objektorientēta programmēšanas valoda.

Galvenās programmas prasību specifikācija

Dokumenta nosaukums: Programmas "AttēluApstrāde" v. 1.0 prasību specifikācija
Dokumenta identifikators: Image.PPS.1.0
Projekta identifikators: Image
Autors: Jevgenijs Jonass

1. Ievads

1.1. Nolūks

Šī dokumenta nolūks ir specificēt programmu "AttēluApstrāde", lai identificētu visas prasības pret programmu, kā arī radītu pamatu detalizētam projektējumam. Programmā netiks realizēta funkcionalitāte, kas nav aprakstīta šajā dokumentā vai saistītajās PPS. Dokumenta mērķauditorija ir programmas izstrādātāji, kas veiks programmas un tās moduļu projektēšanu un programmēšanu.

1.2. Darbības sfēra

Programma"AttēluApstrāde" ir paredzēta fotogrāfiju, teksta zīmju attēlu apstrādei, kvalitātes uzlabošanai, kā arī teksta atpazīšanai un attēlu salīdzināšanai.

1.3. Definīcijas un saīsinājumi

Attēlu apstrāde – attēlos vai citās grafiskajās informācijas atveidošanas formās esošās informācijas analīze, parasti izmantojot signālu ciparapstrādi.
Modulis – atsevišķa identificējama programmas daļa, kuru var autonomi izveidot un izmantot, lai atvieglotu programmu sastādīšanu.
Modāls logs – logs, kas bloķē lietotāja darbu ar galveno logu līdz brīdim, kamēr lietotājs to (modālo logu) neaizvers.
pprv. – pretēji pulksteņa rādītāja virzienam
prv. – pulksteņa rādītāja virzienā
BLSM – bilineāra splaina metode

1.4. Saistība ar citiem dokumentiem

Dokuments tika izstrādāts, balstoties uz standartā LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis" aprakstītajām prasībām. Uz šī dokumenta pamata tiks izstrādāts programmatūras projektējuma apraksts.

Moduļu prasību specifikācijas

- Image.tRec.PPS.1.0 – moduļa textRec prasību specifikācija
- Image.CO.PPS.1.0 – moduļa CompareObj prasību specifikācija
- Image.SPW.PPS.1.0 – moduļa SphericalWave prasību specifikācija
- Image.Rec.PPS.1.0 – moduļa Recognition prasību specifikācija
- Image.Search.PPS.1.0 – moduļa Search prasību specifikācija

2. Vispārējs apraksts

2.1. Programmas moduļi

Moduļa ID	Nosaukums	Apraksts
Main	Main	Galvenais modulis, kas nodrošina pamatfunkcionalitāti, tādu kā attēlu atvēršana, attēlošana un saglabāšana, kā arī satur attēlu apstrādes pamatalgoritmus
CO	CompareObj	Objektu salīdzināšanas modulis
tRec	textRec	Teksta atpazīšanas modulis
Rec	Recognition	Teksta zīmes atpazīšanas modulis
SPW	SphericalWave	Teksta attēlu struktūras analīzes modulis
Search	Search	Līdzīgu attēlu meklēšanas modulis

2.2. Galvenā moduļa funkcijas

2.2.1. Darbošanās ar failiem

Identifikators	Nosaukums
Func.File.1.ImgOpen	Attēla atvēršana
Func.File.2.ImgSave	Attēla saglabāšana
Func.File.3.ImgClose	Attēla aizvēršana

2.2.2. Attēlu rediģēšanas funkcijas

Identifikators	Nosaukums
Func.Edit.1.Gray	Attēla pārveidošana pelēkuma skalā
Func.Edit.2.RotateACW	Attēla pagriešana par 90 grādiem pprv.
Func.Edit.3.RotateCW	Attēla pagriešana par 90 grādiem prv.
Func.Edit.4.HorFlip	Atspguļot horizontāli
Func.Edit.5.VertFlip	Atspguļot vertikāli
Func.Edit.6.Negative	Krāsu inversija
Func.Edit.7.Crop	Attēla apcirpšana
Func.Edit.8.Undo	Darbības atcelšana
Func.Edit.9.Redo	Darbības atkārtošana
Func.Edit.10.frw	Atgriešanās pie sākotnējā attēla
Func.Edit.11.ffw	Atgriešanās pie pēdējā attēla

2.2.3. Attēlu apstrādes algoritmi

Identifikators	Nosaukums
Func.Alg.1.Bernsen	Binarizācija ar Bernsena metodi
Func.Alg.2.Otsu	Sliekšņa (threshold) noteikšana ar Otsu metodi
Func.Alg.3.binthr	Globāla attēla binarizācija, izmantojot noteikto slieksni (threshold)
Func.Alg.4.rsmnn	Attēla izmēru palielināšana ar tuvākā kaimiņa metodi
Func.Alg.5.rsmbil	Attēla izmēru palielināšana/samazināšana ar BLSM
Func.Alg.6.varfilter	Variācijas filtrs
Func.Alg.7.medfilbid	Attēla filtrēšana ar divvirzienu mediānas filtru
Func.Alg.8.medfilmtrx	Attēla filtrēšana ar mediānas filtru ar kvadrātisku matricu
Func.Alg.9.Niblack	Binarizācija ar Nibleka metodi
Func.Alg.10.mnfltrsqr	Attēla filtrēšana ar vidējās vērtības filtru ar kvadrātisku matricu

2.3. Lietotāja raksturiezīmes

Programma paredzēta lietotājiem, kuri grib apstrādāt attēlus. Lietotājam jābūt zināšanām par attēlu apstrādes teoriju, krāsu sintēzi.

2.4. Sistēmas prasības

Programmai jāstrādā *Java* platformā. Visiem tās moduļiem jābūt realizētiem *Java* programmēšanas valodā. Lai programmu lietotu, nepieciešams dators ar sekojošu aparatūru un programmatūru vai to jaunāku versiju:

Windows XP

- Microsoft® Windows® XP with Service Pack 2 (Service Pack 3 recommended) or Windows Vista® Home Premium, Business, Ultimate, or Enterprise with Service Pack 1 (certified for 32-bit Windows XP and 32-bit and 64-bit Windows Vista)
- Procesors 1.8 MHz un vairāk
- Java Virtual Machine
- 100MB operatīvās atmiņas
- Monitora izšķirtspēja 1024x768 (rekomendēta 1280x800) un 16-bitu videokarte

Mac OS

- Procesors PowerPC® G5 vai daudzkodoļu procesors Intel®
- Mac OS X v10.4.11–10.5.4
- 100MB operatīvās atmiņas
- Java for Mac OS X 10.5 Update 2
- Monitora izšķirtspēja 1024x768 (rekomendēta 1280x800) un 16-bitu videokarte

3. Funkcionālās prasības

3.1. Darbošanās ar failiem

3.1.1. Attēla atvēršana

Identifikators	Func.File.1.ImgOpen
Nosaukums	Attēla atvēršana
Mērķis	
Ielādēt vienu attēlu no faila.	
Ievaddati	
Faila vārds.	
Apstrāde	
Attēla dati tiek nolasīti no faila. Ja operētājsistēma atbalsta failu deskriptorus, deskriptors tiek atbrīvots uzreiz pēc faila nolasīšanas (lai failu varētu izdzēst, neaizverot attēlu).	
Izvaddati	
Atvērtais attēls.	

3.1.2. Attēla saglabāšana

Identifikators	Func.File.2.ImgSave
Nosaukums	Attēla saglabāšana
Mērķis	
Saglabāt attēlu failā.	
Ievaddati	
1. Attēls 2. Faila vārds, ieskaitot paplašinājumu (skat. **5.2. Savietojamība**).	
Apstrāde	
1. Tiek iegūts pilns faila nosaukums ar paplašinājumu. 2. Ja fails ar doto nosaukumu eksistē, tas tiek dzēsts. 3. Tiek izveidots jauns fails. 4. Attēls tiek saglabāts failā. 5. Fails tiek aizvērts.	
Izvaddati	
Programma izveido jaunu failu vai pārraksta eksistējošo.	

3.1.3. Attēla aizvēršana

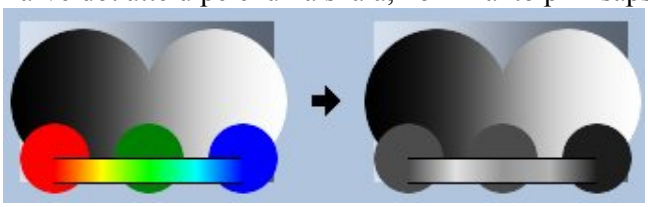
Funkcija domāta, lai lietotājs varētu aizvērt attēlu. Pēc aizvēršanas attēls pazūd no ekrāna.

3.2. Attēlu rediģēšanas funkcijas

3.2.1. Pagriešana un atspoguļošana

Funkcijas *Attēla pagriešana par 90 grādiem pprv.*, *Attēla pagriešana par 90 grādiem prv.*, *Atspguļot horizontāli*, *Atspguļot vertikāli* veic vienkāršu attēla transformāciju. Šīs funkcijas apstrādā attēlu MxN, kas izmanto RGB krāsu modeli, un atgriež attēlu ar izmēru attiecīgi NxM vai MxN.

3.2.2. Attēla pārveidošana pelēkuma skalā

Identifikators	Func.Edit.1.Gray
Nosaukums	Attēla pārveidošana pelēkuma skalā
Mērķis	
Pārveidot attēlu pelēkuma skalā, ko izmanto pirmsapstrādes stadijā. 	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. RGB koeficienti $k_r, k_g, k_b \in \mathbb{Z} \mid 0 \le k_r, k_g, k_b \le 100,\ k_r + k_g + k_b = 100$.	
Apstrāde	
Katram pikselam piešķirt pelēku nokrāsu, kuras intensitāte tiek rēķināta pēc formulas $I(x,y) = \frac{k_r r(x,y) + k_g g(x,y) + k_b b(x,y)}{100}$, kur $r(x,y),\ g(x,y),\ b(x,y)$ – piksela ar koordinātēm (x,y) sarkanā, zaļā un zilā komponente.	
Izvaddati	
Tāda paša izmēra attēls pelēkuma skalā.	

3.2.3. Krāsu inversija

Identifikators	Func.Edit.6.Negative
Nosaukums	Krāsu inversija
Mērķis	
Invertēt attēlu.	
Ievaddati	
Attēls MxN, kas izmanto RGB krāsu modeli.	
Apstrāde	
Funkcija apstrādā katru pikselu un invertē katru RGB komponentu.	
Izvaddati	
Invertēts attēls ar tādu pašu izmēru.	

3.2.4. Attēla apcirpšana

Identifikators	Func.Edit.7.Crop
Nosaukums	Attēla apcirpšana
Mērķis	
Grafiska attēla rediģēšana, kurā apgriež attēla malas vai atmet nevajadzīgās daļas, lai to varētu ievietot atvēlētajā displeja ekrāna laukumā.	
Ievaddati	
1. Attēls MxN, kas izmanto RGB krāsu modeli. 2. Attēla taisnstūra apgabals KxL. Apgabala izvēle notiek ar peles palīdzību (skat. **4.2. Galvenais logs**).	
Apstrāde	
Funkcija izveido jaunu attēlu KxL un kopē pikselus no izvēlētā apgabala uz jauno attēlu.	
Izvaddati	
Apcirptais attēla taisnstūra apgabals KxL.	

3.2.5. Darbības atcelšana/atkārtošana

Komandas *Darbības atcelšana*, *Darbības atkārtošana*, *Atgriešanās pie sākotnējā attēla*, *Atgriešanās pie pēdējā attēla* paredzētas, lai lietotājs varētu atcelt vai atkārtot darbību. Šīs 4 komandas attiecas uz attēlu, kas redzams galvenajā logā, un tā izmaiņām. Fukcijas, kuru darbības rezultātā attēls mainās, var atcelt. Pēc attēla aizvēršanas visas veiktās izmaiņas tiek pazaudētas.

3.3. Attēlu apstrādes algoritmi

3.3.1. Binarizācija ar Bernsena metodi

Identifikators	Func.Alg.1.Bernsen
Nosaukums	Binarizācija ar Bernsena metodi
Mērķis	
Binarizēt attēlu, atdalot objektus no fona.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. RGB koeficienti $k_r, k_g, k_b \in \mathbb{Z} \mid 0 \le k_r, k_g, k_b \le 100,\ k_r + k_g + k_b = 100$. 3. radiuss $r \in \mathbb{Z}_{\ge 0}$ 4. kontrasta slieksnis (threshold) – vesels skaitlis $t \in [0..255]$	
Apstrāde	
Funkcija katram attēla pikselam meklē kontrastu, kas tiek rēķināts kā starpība starp maksimālo un minimālo intensitāti kvadrātveida apgabalā ar radiusu r, un salīdzina to ar slieksni. Ja kontrasts ir pietiekami liels, piksels kļūst melns, pretējā gadījumā balts.	
Izvaddati	
Melnbalts attēls ar tādu pašu izmēru.	

3.3.2. Sliekšņa (threshold) noteikšana ar Otsu metodi

Identifikators	Func.Alg.2.Otsu
Nosaukums	Sliekšņa (threshold) noteikšana ar Otsu metodi
Mērķis	
Izrēķināt optimālu binarizācijas sliekšņa (threshold) vērtību.	
Ievaddati	
1) Rastra attēls, kas izmanto RGB krāsu modeli 2) RGB koeficienti $k_r, k_g, k_b \in \mathbb{Z} \mid 0 \le k_r, k_g, k_b \le 100,\ k_r + k_g + k_b = 100$.	
Apstrāde	
Atgriezt globālās binarizācijas sliekšņa vērtību, kas atbilst maksimālajai intensitāšu klašu dispersijai, kas tiek rēķināta kā $\delta^2 = \omega_0(k)\omega_1(k)\left(\mu_0(k) - \mu_1(k)\right)^2$, kur $\omega_0(k), \omega_1(k)$ ir klašu svērtās vidējās vērtības (respective means), $\mu_0(k), \mu_1(k)$ ir katras klases vidēji intensitātes līmeņi, kas tiek rēķināti pēc formulas: $\omega_0(k) = \sum_{i=1}^{k} p_i \qquad \mu_0(k) = \sum_{i=1}^{k} \frac{ip_i}{\omega_0(k)}$ $\omega_1(k) = \sum_{i=k+1}^{L} p_i = 1 - \omega_0(k) \qquad \mu_1(k) = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1(k)}$ kur p_i ir intensitātes i biežums: $p_i = \frac{N_i}{N}$, kur N ir kopējais pikselu skaits	
Izvaddati	
Vesels skaitlis diapazonā [0..255] – optimālais slieksnis (threshold)	

3.3.3. Globāla attēla binarizācija, izmantojot noteikto slieksni (threshold)

Identifikators	Func.Alg.3.binthr
Nosaukums	Globāla attēla binarizācija, izmantojot noteikto slieksni (threshold)
Mērķis	
Binarizēt krāsainu attēlu.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. slieksnis (threshold) – vesels skaitlis $t \in [0..255]$ 3. RGB koeficienti $k_r, k_g, k_b \in \mathbb{R} \mid 0 \le k_r, k_g, k_b \le 100,\ k_r + k_g + k_b = 100$. Ievade notiek binarizācijas logā (skat. **4.3. Attēla binarizācijas logs**)	
Apstrāde	
Katru pikselu tiek pārveidot par baltu, ja tā intensitāte ir pietiekami liela, vai par melnu, ja intensitāte ir pārāk zema. Intensitātes lielums tiek noteikts, salīdzinot to ar slieksni.	
Izvaddati	
Melnbalts attēls ar tādu pašu izmēru.	

3.3.4. Attēla izmēru palielināšana ar tuvākā kaimiņa metodi

Identifikators	Func.Alg.4.rsmnn
Nosaukums	Attēla izmēru palielināšana ar tuvākā kaimiņa metodi
Mērķis	
Palielināt attēlu ar ievērojamiem kvalitātes zaudējumiem.	
Ievaddati	
1. Rastra attēls $A(M \times N)$, kas izmanto RGB krāsu modeli 2. izstiepšanas koeficienti k_x un k_y – veseli skaitļi ($k_x, k_y > 0$)	
Apstrāde	
Izveido attēlu, kurš tiek izstiepts, palielinot katru attēla punktu un neveicot rezultāta gludināšanu.	
Izvaddati	
Palielinātais attēls $B(k_x M \times k_y N)$	

3.3.5. Attēla izmēru palielināšana/samazināšana ar BLSM

Identifikators	Func.Alg.5.rsmbil
Nosaukums	Attēla izmēru palielināšana/samazināšana ar BLSM
Mērķis	
Palielināt attēlu, pēc iespējas mazāk zaudējot kvalitāti.	
Ievaddati	
1. Rastra attēls $A(M \times N)$, kas izmanto RGB krāsu modeli 2. Jauns izmērs $K \times L$ $(K, L \in \mathbb{Z}_{>1})$ vai izstiepšanas koeficienti $k_x, k_y \in \mathbb{Z}_{>0}$	
Apstrāde	
Izrēķināt katra piksela vērtību, interpolējot to pēc 3 blakus esošiem pikseliem un tekošā piksela. Piksela pozīcija ieejas attēlā tiek rēķināta, izmantojot mērogošanas koeficientus, kas tiek iegūti, izdalot jaunā attēla izmērus ar vecā attēla izmēriem.	
Izvaddati	
Palielinātais/samazinātais attēls $B(K \times L)$	

3.3.6. Variācijas filtrs

Identifikators	Func.Alg.6.varfilter
Nosaukums	Variācijas filtrs
Mērķis	
Nofiltrēt attēlu, meklējot apgabalus, kuros sastopama vislielāka intensitāšu variācija.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. Fitrēšanas radiuss $r \in \mathbb{Z}_{\geq 0}$ 3. slieksnis (threshold) – vesels skaitlis $t \in [0..255]$ 4. Attēla taisnstūra apgabals.	
Apstrāde	
Katram pikselam A(x;y) piešķirt intensitāti, kas tiek rēķināta kā $\left(\sum_{i=x-r}^{x+r} \sum_{j=y-r}^{y+r} \left(A(x;y) \right)^2 \right) - M^2$, kur M ir intensitātes vidējā vērtība kvadrātiskā apgabalā ar radiusu r. Piksela intensitāte tiek mainīta tikai tad, ja starpība starp veco un jauno vērtību lielāka par slieksni.	
Izvaddati	
Filtrēts attēls ar tādu pašu izmēru.	

3.3.7. Attēla filtrēšana ar divvirzienu mediānas filtru

Identifikators	Func.Alg.7.medfilbid
Nosaukums	Attēla filtrēšana ar divvirzienu mediānas filtru
Mērķis	
Nofilterēt trokšņus attēlā.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. Fitrēšanas radiuss $r \in \mathbb{Z}_{\geq 0}$ 3. slieksnis (threshold) – vesels skaitlis $t \in [0..255]$ 4. Attēla taisnstūra apgabals.	
Apstrāde	
Attēlu apstrādā divreiz: vispirms horizontāli, pēc tam vertikāli. Horizontālā apstrāde: katru pikselu aizvietot ar mediānu no tajā pašā rindā esošajiem 2r+1 tuvākajiem pikseliem. Vertikālā apstrāde: katru pikselu aizvietot ar mediānu no tajā pašā kolonnā esošajiem 2r+1 tuvākajiem pikseliem. Piksels tiek aizvietots ar mediānu tikai tad, ja starpība starp veco un jauno vērtību lielāka par slieksni.	
Izvaddati	
Filtrēts attēls ar tādu pašu izmēru.	

3.3.8. Attēla filtrēšana ar mediānas filtru ar kvadrātisku matricu

Identifikators	Func.Alg.8.medfilmtrx
Nosaukums	Attēla filtrēšana ar mediānas filtru ar kvadrātisku matricu
Mērķis	
Nofilterēt trokšņus attēlā.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. Fitrēšanas radiuss $r \in \mathbb{Z}_{\geq 0}$ 3. slieksnis (threshold) – vesels skaitlis $t \in [0..255]$ 4. Attēla taisnstūra apgabals.	
Apstrāde	
Katram pikselam piešķirt intensitāti, kas tiek rēķināta kā mediāna no kvadrātiskā apgabalā ar radiusu r esošo pikselu intensitātēm. Piksels tiek aizvietots ar mediānu tikai tad, ja starpība starp veco un jauno intensitātes vērtību lielāka par slieksni.	
Izvaddati	
Filtrēts attēls ar tādu pašu izmēru.	

3.3.9. Binarizācija ar Nibleka metodi

Identifikators	Func.Alg.9.Niblack
Nosaukums	Binarizācija ar Nibleka metodi
Mērķis	
Binarizēt attēlu, atdalot objektus no fona.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. RGB koeficienti $k_r, k_g, k_b \in \mathbb{Q} \mid 0 \le k_r, k_g, k_b \le 100,\ k_r + k_g + k_b = 100$. 3. radiuss $r \in \mathbb{Q}_{\ge 0}$ 4. koeficients $k \in \mathbb{Q}$	
Apstrāde	
Katra piksela intensitāte tiek salīdzināta ar lokālo slieksni, ko iegūst kā $B(x,y) = m(x,y) + ks(x,y)$, kur $m(x,y), s(x,y)$ ir vidējā vērtība un standarta dispersija iztvērumā ar radiusu r. Ja intensitāte lielāka par slieksni, piksels kļūst balts, pretējā gadījumā – melns.	
Izvaddati	
Melnbalts attēls ar tādu pašu izmēru.	

3.3.10. Attēla filtrēšana ar vidējās vērtības filtru ar kvadrātisku matricu

Identifikators	Func.Alg.10.mnfltrsqr
Nosaukums	Attēla filtrēšana ar vidējās vērtības filtru ar kvadrātisku matricu
Mērķis	
Padarīt attēlu gludāku, samazinot intensitāšu līmeņu variāciju.	
Ievaddati	
1. Rastra attēls, kas izmanto RGB krāsu modeli 2. Fitrēšanas radiuss $r \in \mathbb{Q}_{\ge 0}$ 3. slieksnis (threshold) – vesels skaitlis $t \in [0..255]$ 4. Attēla taisnstūra apgabals.	
Apstrāde	
Katram pikselam piešķirt intensitāti, kas tiek rēķināta kā vidējā vērtība no kvadrātiskā apgabalā ar radiusu r esošo pikselu intensitātēm. Piksels tiek aizvietots ar vidējo vērtību tikai tad, ja starpība starp veco un jauno intensitātes vērtību lielāka par slieksni.	
Izvaddati	
Filtrēts attēls ar tādu pašu izmēru.	

4. Grafiskā saskarne

Programmas saskarne sastāv no daudziem neatkarīgiem logiem, kas tiek izsaukti, izmantojot attiecīgus izvēlnes punktus. Saskarnei jābūt angļu valodā. Katram saskarnes komponentam jābūt atspējotam, ja funkciju, par kuru tas atbild, nedrīkst izsaukt dotajā brīdī. Visos logos, kuros tiek parādīts attēls, jāparedz gadījums, kad attēls neietilpst logā. Jādod iespēja ritināt attēlu ar ritjoslu palīdzību.

4.1. Logu uzskatījums

Nosaukums	Kur aprakstīts
Galvenais logs	Šajā dokumetā, sadaļā 4.2
Attēla binarizācijas logs	Šajā dokumetā, sadaļā 4.3
Filtrēšanas logs	Šajā dokumetā, sadaļā 4.4
Attēla izmēra mainīšanas logs	Šajā dokumetā, sadaļā 4.5
Attēla saglabāšanas logs	Šajā dokumetā, sadaļā 4.6
Attēla atvēršanas logs	Šajā dokumetā, sadaļā 4.6
Attēla struktūras analīzes logs	Image.SPW.PPS.1.0, sadaļā 4.1
Teksta zīmes atpazīšanas logs	Image.Rec.PPS.1.0, sadaļā 2.4
Teksta zīmes pievienošanas datu bāzei logs	Image.Rec.PPS.1.0, sadaļā 2.4
Objektu salīdzināšanas logs	Image.CO.PPS.1.0, sadaļā 2.3.3

4.2. Galvenais logs

Galvenais logs atveras pēc programmas palaišanas. Tas sastāv no paneles, kurā parādās ielādētais attēls, un izvēlnes, zem kuras ir novietota rīkjosla. Lietotājam jāļauj izvēlēties attēla apgabalu ar peles palīdzību, nospiežot uz attēla un pārvietojot peles kursoru.

Izvēlnes punkti

Nosaukums	Apraksts
File	Failu atvēršana un aizvēršana
Edit	Attēlu rediģēšana
Tools	Attēlu apstrādes algoritmi
Recognition	Attēlā redzamās teksta zīmes atpazīšana (aprakstīts dokumentā Image.Rec.PPS.1.0)
Compare	Objektu salīdzināšana (aprakstīts dokumentā Image.CO.PPS.1.0)
Search	Līdzīgu attēlu meklēšana (aprakstīts dokumentā Image.Search.PPS.1.0)

Izvēlnes punkts *File*

Izvēlnes komanda	Izsaucamās funkcijas identifikators
Open	Func.File.1.ImgOpen
Save as	Func.File.2.ImgSave
Close	Func.File.3.ImgClose
Quit	– (programmas aizvēršana)

Izvēlnes punkts *Edit*

Izvēlnes komanda	Izsaucamās funkcijas identifikators	Parametri
Undo	Func.Edit.8.Undo	
Redo	Func.Edit.9.Redo	
Crop	Func.Edit.7.Crop	
Grey image	Func.Edit.1.Gray	30, 59, 11
Rotate acw	Func.Edit.2.RotateACW	
Rotate cw	Func.Edit.3.RotateCW	
Horizontal flip	Func.Edit.4.HorFlip	
Vertical flip	Func.Edit.5.VertFlip	
Negative	Func.Edit.6.Negative	
Strectch horizontally	Func.Alg.4.rsmnn	2, 1
Strectch vertically	Func.Alg.4.rsmnn	1, 2

Izvēlnes punkts *Tools*

Izvēlnes komanda	Izsaucamais logs
Binarization	Attēla binarizācijas logs
Resize	Attēla izmēra mainīšanas logs
Spherical wave	Attēla struktūras analīzes logs (aprakstīts Image.SPW.PPS.1.0)
Filter	Filtrēšanas logs

Rīkjosla

Rīkjoslai jāiekļauj pogas šādu funkciju izsaukšanai:

Nosaukums	Izsaucamās funkcijas identifikators	Parametri
Attēla ielādēšana	Func.File.1.ImgOpen	
Attēla saglabāšana	Func.File.2.ImgSave	
Attēla aizvēršana	Func.File.3.ImgClose	
Atgriešanās pie sākotnējā attēla	Func.Edit.10.frw	
Darbības atcelšana	Func.Edit.8.Undo	
Darbības atkārtošana	Func.Edit.9.Redo	
Atgriešanās pie pēdējā attēla	Func.Edit.11.ffw	
Attēla pagriešana par 90 grādiem pprv.	Func.Edit.2.RotateACW	
Attēla pagriešana par 90 grādiem prv.	Func.Edit.3.RotateCW	
Attēla palielināšana/samazināšana ar BLSM	Func.Alg.5.rsmbil	(67%, 67%)
Attēla palielināšana/samazināšana ar BLSM		(150%, 150%)

4.3. Attēla binarizācijas logs

Neatkarīgs, pārvietojams, nemodāls logs ar maināmu izmēru, kas ļauj lietotājam pārvērst attēlu par melnbaltu, pielietojot dažādas metodes, kā arī pārveidot attēlu pelēkuma skalā. Pēc loga atvēršanas tajā parādās attēla kopija no galvenā programmas loga. Pēc loga atvēršanas tajā parādās attēla kopija no galvenā programmas loga. Metodes tiek pielietotas visam attēlam, nevis tā daļai, un izmaiņas neattiecas uz attēlu galvenajā logā, kamēr netiks nospiesta poga *OK*.

Saskarnes komponents	Funkcija
panele	parāda ielādēto attēlu
poga *OK*	binarizēts attēls parādās galvenajā programmas logā; logs tiek aizvērts
poga *Close*	visas veiktās izmaiņas tiek pazaudētas; logs tiek aizvērts
poga *Otsu*	Func.Alg.2.Otsu
poga *Default*	Uzstāda rgb vērtības pēc noklusējuma: 30, 59, 11
2 pogas *Apply*	Func.Alg.1.Bernsen, Func.Alg.9.Niblack
4 ritlodziņi	ļauj ievadīt slieksni (threshold) un RGB koeficientus
4 teksta lodziņi	ļauj ievadīt precīzas sliekšņa (threshold) un RGB koeficientu vērtības
4 teksta lodziņi	ļauj ievadīt funkciju Func.Alg.1.Bernsen un Func.Alg.9.Niblack parametrus
4 radiopogas	ļauj izvēlēties algoritmu: Func.Alg.1.Bernsen, Func.Alg.9.Niblack, Func.Alg.3.binthr vai Func.Edit.1.Gray

4.4. Filtrēšanas logs

Neatkarīgs, pārvietojams, nemodāls logs ar maināmu izmēru, kas ļauj lietotājam pielietot dažādus filtrus, ievadot to parametrus. Pēc loga atvēršanas tajā parādās attēla kopija no galvenā programmas loga. Lietotājam jāļauj izvēlēties attēla apgabalu ar peles palīdzību, nospiežot uz attēla un pārvietojot peles kursoru. Filtrs tiek pielietots izvēlētajam apgabalam, bet ja apgabals nav izvēlēts, tad visam attēlam. Izmaiņas neattiecas uz attēlu galvenajā logā, kamēr netiks nospiesta poga *OK*.

Saskarnes komponents	Funkcija
panele	parāda ielādēto attēlu
poga *OK*	filtrēts attēls parādās galvenajā programmas logā; logs tiek aizvērts
poga *Original*	tiek parādīts oriģinālais attēls
poga *Close*	visas veiktās izmaiņas tiek pazaudētas; logs tiek aizvērts
poga *Filter*	Attēlam tiek pielietots izvēlētais filtrs
2 teksta lauki	ļauj ievadīt radiusu un slieksni (threshold)
4 radiopogas	ļauj izvēlēties filtra veidu: Func.Alg.7.medfilbid, Func.Alg.8.medfilmtrx, Func.Alg.10.mnfltrsqr vai Func.Alg.6.varfilter

4.5. Attēla izmēra mainīšanas logs

Neatkarīgs, pārvietojams, nemodāls fiksēta izmēra logs, kas ļauj lietotājam mainīt attēla izmēru.

Komponents	Funkcija
2 radiopogas	ļauj izvēlēties algoritmu: Func.Alg.4.rsmnn vai Func.Alg.5.rsmbil
poga *Hide*	logs tiek paslēpts
poga *Resample*	attēlam galvenajā logā tiek pielietots izvēlētais algoritms
5 teksta lauki	ļauj ievadīt jaunu augstumu un plašumu vai izstiepšanas koeficientus; kā arī parāda tekošo izmēru

4.6. Faila atvēršanas un saglabāšanas logi

Šie logi paredzēti attēla ielādēšanai vai saglabāšani failā, izmantojot lietotāja saskarni. Logos jāparedz viegla navigācija failu sistēmā. Saglabāšanas logam jānodrošina iespēja izvēlēties faila formātu. Saglabāšanas logs paredz sekojošus kļūdu paziņojumus:

- "Error occurred while writing to file."

Atvēršanas logs paredz sekojošus kļūdu paziņojumus:

- "Error occurred while reading from file."
- "Image too small." – ja nu gadījumā attēla izmērs mazāks par 1x1.
- "Image too large." – ja attēls ir pārāk liels (skat. **5.1. Veiktspēja**).

5. Nefunkcionālās prasības

5.1. Veiktspēja

Šajā programmas versijā (1.0) netiek izvirzītas nekādas prasības pret ātrdarbību. Tomēr tiek izvirzītas prasības pret attēlu izmēriem: programmai jāspēj atvērt, apstrādāt un saglabāt attēlus ar izmēriem līdz 1280x1280. Programma aizliedz atvērt lielākus attēlus, kā arī aizliedz mainīt attēla izmēru, ja jaunais izmērs ir pārāk liels.

5.2. Savietojamība

Programmai jāspēj atvērt un saglabāt attēlus sekojošos formātos: jpg, jpeg, gif, bmp, png. Programmai jāapstrādā attēli, kas izmanto RGB krāsu modeli. No programmas netiek prasīts citu krāsu modeļu atbalsts. Programma neatbalsta arī *aplha*[1] kanāla informāciju. Precīzāk, programma spēj atvērt attēlus ar *alpha* kanālu, bet *alpha* kanāla informācija būs pazaudēta.

[1] Alpha kanāls atbild par caurspīdīgumu: http://en.wikipedia.org/wiki/Alpha_compositing

Galvenās programmas projektējuma apraksts

Dokumenta nosaukums: Programmas „AttēluApstrāde” v. 1.0 Programmatūras projektējuma apraksts
Dokumenta identifikators: Image.PPA.1.0
Projekta identifikators: Image
Autors: Jevgenijs Jonass

1. Ievads

1.1. Nolūks

Šī dokumenta nolūks ir aprakstīt programmas "AttēluApstrāde" projektējumu, pamatā aprakstot tās galveno moduli (*Main*). Dokumentā netiks precīzi aprakstītas visas metodes, jo tam ir paredzēts *Javadoc* dokumentēšanas rīks. Dokumenta mērķauditorija ir programmas izstrādātāji.

1.2. Darbības sfēra

Programma"AttēluApstrāde" ir paredzēta fotogrāfiju, teksta zīmju attēlu apstrādei, kvalitātes uzlabošanai, kā arī teksta atpazīšanai un attēlu salīdzināšanai.

1.3. Definīcijas un saīsinājumi

Attēlu apstrāde – attēlos vai citās grafiskajās informācijas atveidošanas formās esošās informācijas analīze, parasti izmantojot signālu ciparapstrādi.
Modulis – atsevišķa identificējama programmas daļa, kuru var autonomi izveidot un izmantot, lai atvieglotu programmu sastādīšanu.
Modāls logs – logs, kas bloķē lietotāja darbu ar galveno logu līdz brīdim, kamēr lietotājs to (modālo logu) neaizvers.
pprv. – pretēji pulksteņa rādītāja virzienam
prv. – pulksteņa rādītāja virzienā
BLSM – bilineāra splaina metode

1.4. Saistība ar citiem dokumentiem

Dokuments tika izstrādāts, balstoties uz standartā LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai” aprakstītajām prasībām.

Moduļu projektējuma apraksti

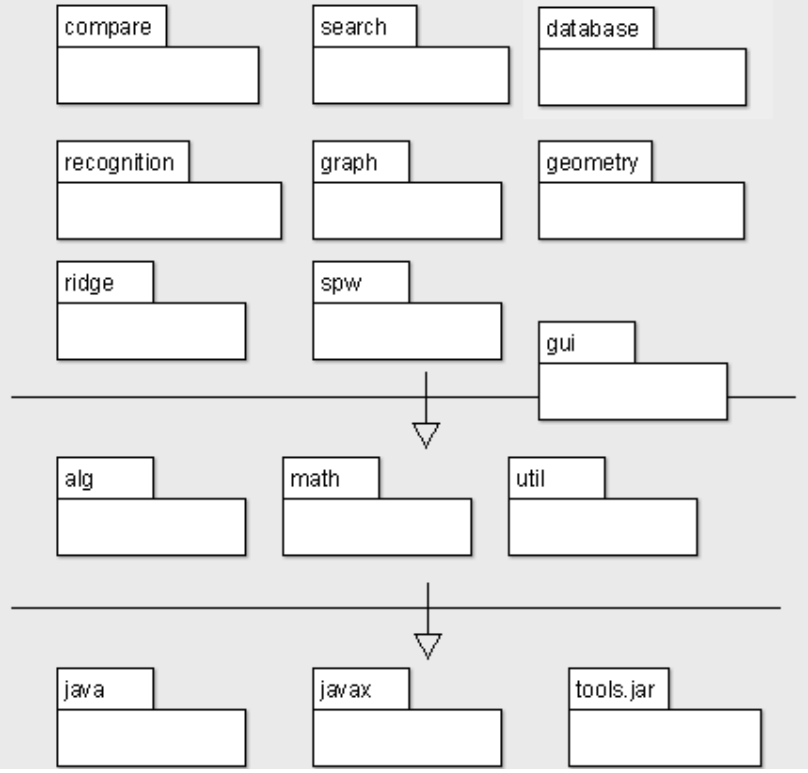
- Image.PPS.1.0 – programmas "AttēluApstrāde" prasību specifikācija
- Image.tRec.PPA.1.0 – moduļa textRec projektējuma apraksts
- Image.CO.PPA.1.0 – moduļa CompareObj projektējuma apraksts
- Image.SPW.PPA.1.0 – moduļa SphericalWave projektējuma apraksts
- Image.Rec.PPA.1.0 – moduļa Recognition projektējuma apraksts
- Image.Search.PPA.1.0 – moduļa Search projektējuma apraksts

Šī dokumeta sastāvdaļa ir projekta "AttēluApstrāde" *Javadoc* dokumentācija.

2. Dekompozīcijas apraksts

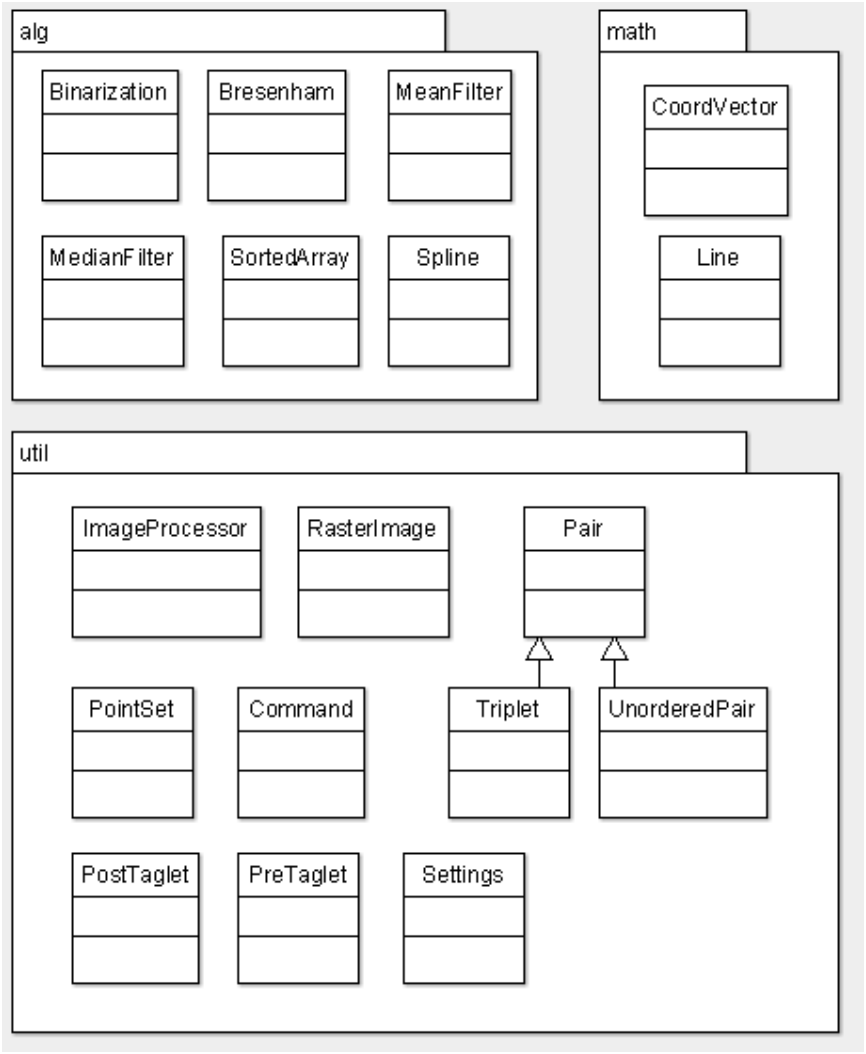
2.1. Moduļu dekompozīcija

Programma sastāv no vairākiem moduļiem, no kuriem viens ir galvenais (*Main*). Katrs modulis sastāv no vienas vai vairākām *Java* pakotnēm. Sekojoša diagramma parāda moduļu dekompozīciju:

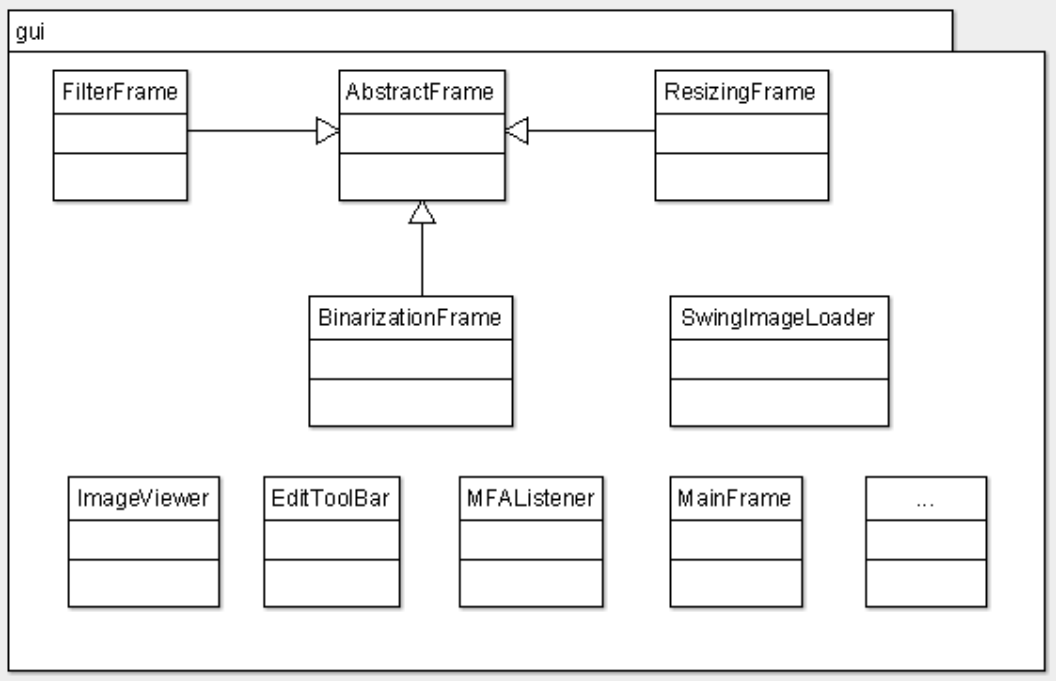


Paskaidrojumi: centrā ir 3 pakotnes, kas ietilpst galvenā moduļa sastāvā: *alg*, *math*, *util*. Pakotne *gui* satur galvenā moduļa un citu moduļu grafiskos komponentus. Diagrammas augšā ir pakotnes, kas ietilpst citu moduļu sastāvā. Apakšā ir *Java* valodas standartpakotnes.

2.2. Galvenā moduļa dekompozīcija



2.3. Grafiskās saskarnes komponenti



Galvenā moduļa grafiskās saskarnes komponenti

2.4. Pakotņu un klašu apraksts

Pakotne	Apraksts
alg	Attēlu apstrādes pamatalgoritmi.
math	Palīgklases matemātiskajiem aprēķiniem.
util	Palīgklases.
gui	Grafiskās saskarnes komponenti.

Pakotne *alg*

Klase	Apraksts
Spline	Klases metodes realizē splaina algoritmus attēla izmēra mainīšanai.
Binarization	Klases metodes realizē attēla binarizācijas algoritmus.
Bresenham	Klases metodes realizē Bresenhama algoritmu.
MedianFilter	Klases metodes realizē mediānas filtru.
MeanFilter	Klases metodes realizē vidējās vērtības un variācijas filtrus.
SortedArray	Klase reprezentē sakārtotu masīvu.

Pakotne *gui*

Klase	Apraksts
AbstractFrame	Abstrakta klase, kas satur kopīgas metodes daudziem logiem.
MainFrame	Galvenā klase, kas reprezentē galveno logu un satur metodi *main*.
MFAListener	Klase, kas paredzēta galvenā loga notikumu apstrādei.
FilterFrame	Logs, kas atbild par attēla filtrēšanu.
BinarizationFrame	Logs, kas atbild par binarizāciju.
ResizingFrame	Logs, kas ļauj mainīt attēla izmērus.
ImageViewer	Panele, kura parāda ielādēto attēlu un tiek izmantota daudzos logos.
EditToolBar	Galvenā loga rīkjosla.
SwingImageLoader	Klase ļauj saglabāt/atvērt attēlu no faila.
...	Citu moduļu logi, kas aprakstīti attiecīgajos PPA/PPS

Pakotne *util*

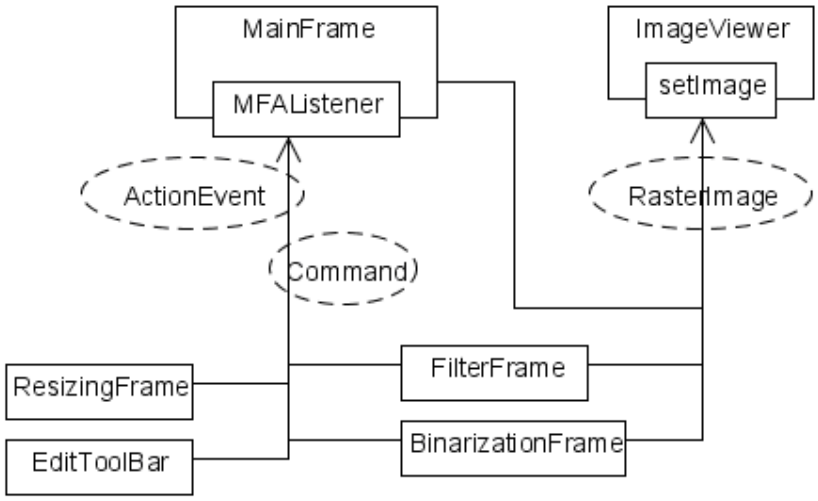
Klase	Apraksts
ImageProcessor	Klase satur attēlu apstrādes pamatmetodes.
RasterImage	Klase reprezentē rastra attēlu.
Pair	Klase reprezentē sakārtotu elementu pāri.
Triplet	Klase reprezentē sakārtotu elementu trijnieku.
UnorderedPair	Klase reprezentē nesakārtotu elementu pāri.
PointSet	Klase-alias, kas ekvivalenta klasei HashSet <Point>. Ieviesta ērtības dēļ.
Command	Klase reprezentē komandu ar parametriem galvenajam logam.
PostTaglet	Klases izmantotas javadoc dokumentācijas ģenerēšanai.
PreTaglet	
Settings	Abstrakta klase, kas reprezentē algoritmu parametrus.

Pakotne *math*

Klase	Apraksts
CoordVector	Reprezentē koordināšu vektoru ar *double* tipa elementiem.
Line	Satur metodes matemātiskajiem aprēķiniem, kas saistīti ar taisnēm.

3. Projektējuma apraksts

3.1. Grafisko komponenšu savstarpējā saskarne



3.2. MFAListener komandas

Komandas ar tipu ActionEvent nāk no galvenā loga izvēlnes un no rīkjoslas, bet komandas ar tipu Command nāk no rīkjoslas un citiem logiem un satur papildus parametrus.

Komandas ar tipu *Command*

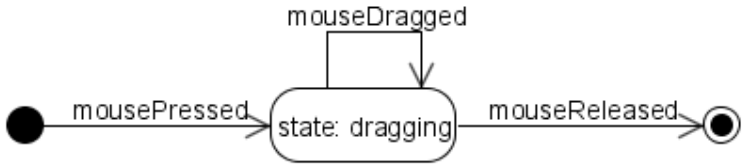
Komanda	"Bilinear resample"	
Avots (source)	ResizingFrame, EditToolBar	
Apraksts	Komanda tiek sūtīta, kad lietotājs maina attēla izmēru ar BLSM metodi.	
Parametru skaits	3	
Parametri		
Parametrs	Tips	Apraksts
args[0]	Integer	režīms: 1 – komanda satur jaunu attēla izmēru; 2 – komanda satur izstiepšanas koeficientus
args[1]	ja args[0]=1: Integer ja args[1]=2: Double	ja args[0]=1: jaunais platums ja args[1]=2: horizontālās izstiepšanas koeficients
args[2]	ja args[0]=1: Integer ja args[1]=2: Double	ja args[0]=1: jaunais augstums ja args[1]=2: vertikālās izstiepšanas koeficients

Komanda	"NN resample"	
Avots (source)	ResizingFrame	
Apraksts	Komanda tiek sūtīta, kad lietotājs maina attēla izmēru ar tuvākā kaimiņa metodi.	
Parametru skaits	2	
Parametri		
Parametrs	Tips	Apraksts
args[0]	Integer	horizontālās izstiepšanas koeficients
args[1]	Integer	vertikālās izstiepšanas koeficients

Komanda	"Change image"	
Parametru skaits	1	
Apraksts	Komanda tiek sūtīta, kad lietotājs nospied pogu *OK* binarizācijas vai filtrēšanas logā.	
Avots (source)	BinarizationFrame, FilterFrame	
Parametri		
Parametrs	Tips	Apraksts
args[0]	RasterImage	Norāde uz attēlu, kas jāattēlo galvenajā logā.

3.3. Attēla apgabala izvēle

Attēla apgabala izvēle ir process, kurā lietotājs ar peles palīdzību nospied uz apgabala stūra punkta un pārvieto peles kursoru uz diagonāli pretējo stūru. Šo procesu modelē galīgs automāts:



Attēla apgabala izvēle realizēta klasē *ImageViewer*, kura atbild par attēlu rādīšanu, kā arī par lietotāju ievades apstrādi, kas veikta ar peles palīdzību. Klasei ir metode `public Rectangle getSelection()` izvēlētā apgabala noteikšanai, ko izmanto grafisko logu klases. Galīgs automāts realizēts ar 3 privāto lauku un 2 privātu klašu palīdzību:

- `private Point start, end;` – sākuma un beigu kursora pārvietošanas punkti
- `private Rectangle selection; – izvēlētais apgabals`
- `private class MListener implements MouseListener {...}`
- `private class MMotionListener implements MouseListener {...}`

Visu vajadzīgo peles notikumu apstrāde notiek ar sekojošām metodēm:

- public void MListener.mousePressed(MouseEvent e)
- public void MListener.mouseReleased(MouseEvent e)
- public void MMotionListener.mouseDragged(MouseEvent e)

Metodes apraksta sekojošais pseidokods (currentPos apzīmē tekošo kursora pozīciju):

```
mousePressed {
        start=currentPos
}
mouseReleased {
        end=currentPos
        selection=rectangle(start.x, start.y, end.x, end.y)
}
mouseDragged {
        end=currentPos
}
```

3.4. Grafisko logu projektējums

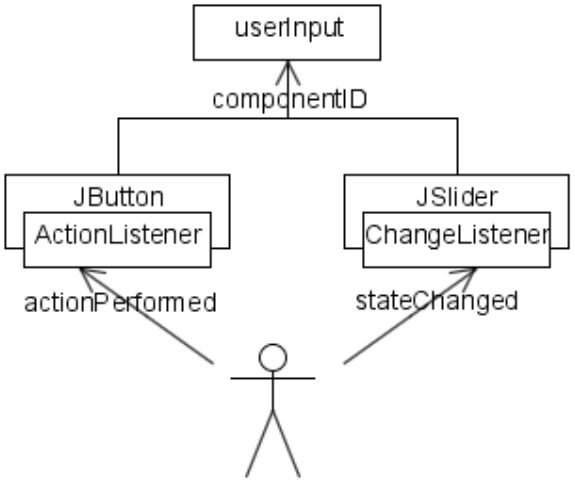
Grafisko saskarni nodrošina *Swing* komponenti. Grafisko logu reprezentē abstrakta klase *AbstractFrame*, kas satur metodes, kuras ir kopīgas visiem logiem un kalpo loģikas atdalīšanai no saskarnes:

1. `createPanels()` – izveido un izvieto paneles (*JPanel* tipa objektus)
2. `createComponents()` – izveido un formatē *Swing* komponentus
3. `addComponents()` – pievieno komponentus panelēm
4. `createListeners()` – izveido komponenšu notikumu klausītājus
5. `addListeners()` – pievieno komponentiem notikumu klausītājus
6. `enableInput()` – uzstāda komponenšu statusu uz enabled/disabled
7. `userInput(int componentID)`
8. `createAndShowGUI()`

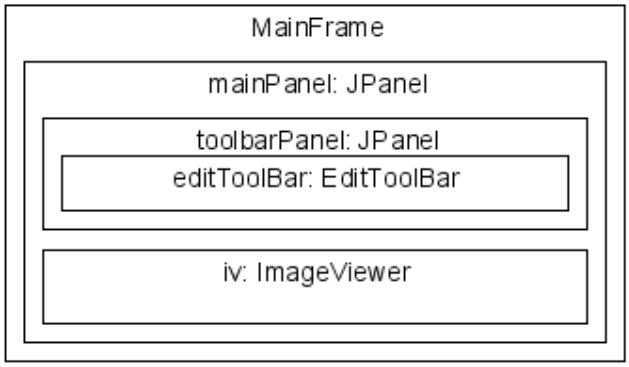
Metodes 1–7 ir ar modifikatoriem `abstract, protected` un ar tipu `void`. Metode `createAndShowGUI`() ir jāizsauc visu grafisko logu konstruktoros:

```
protected final void createAndShowGUI() {
      createPanels();
      createComponents();
      addComponents();
      createListeners();
      addListeners();
      enableInput();
      setVisible(true);
}
```

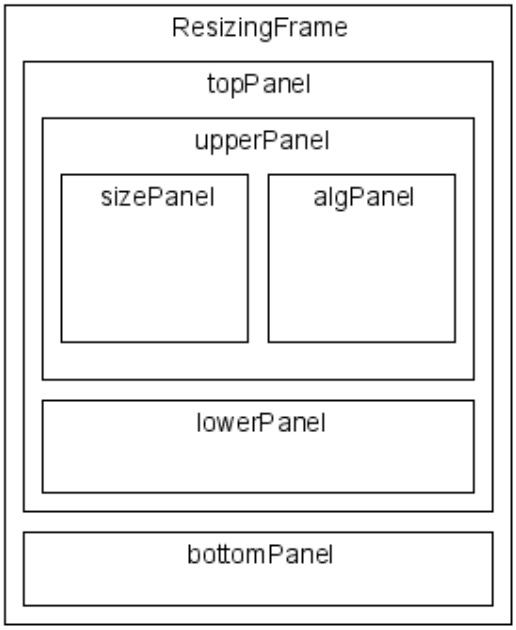
Katram komponentam, ar kura palīdzību lietotājs veic datu ievadi, tiek piešķirts identifikators loga klases ietvaros, kuru saņem metode `userInput`, kas kalpo lietotāja ievades apstrādei. Visi ievades notikumi tiek pāradresēti uz šo metodi:



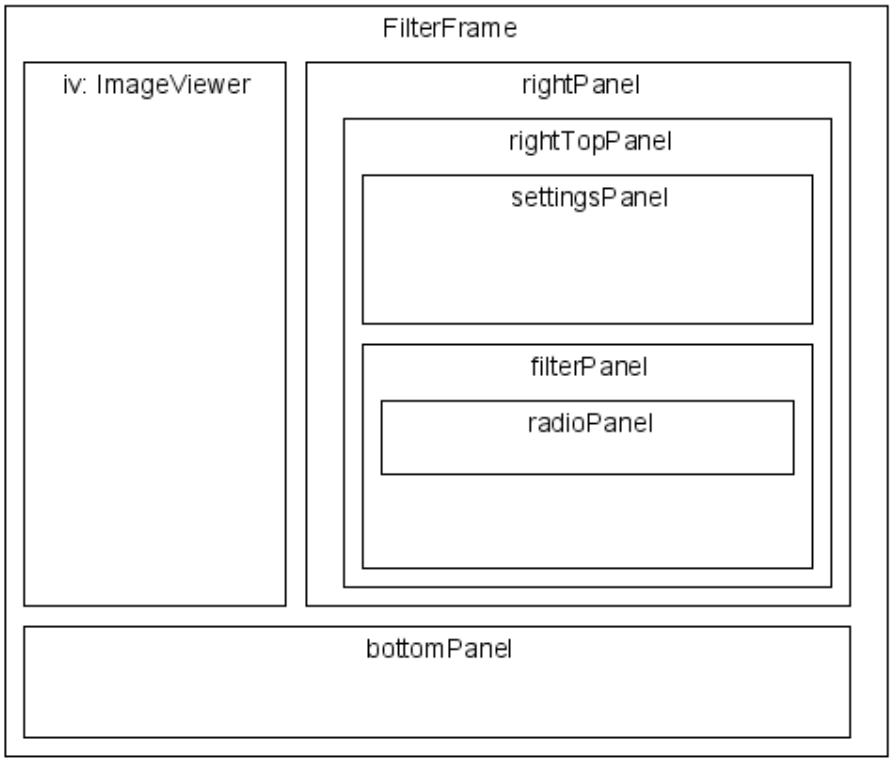
3.4.1. MainFrame



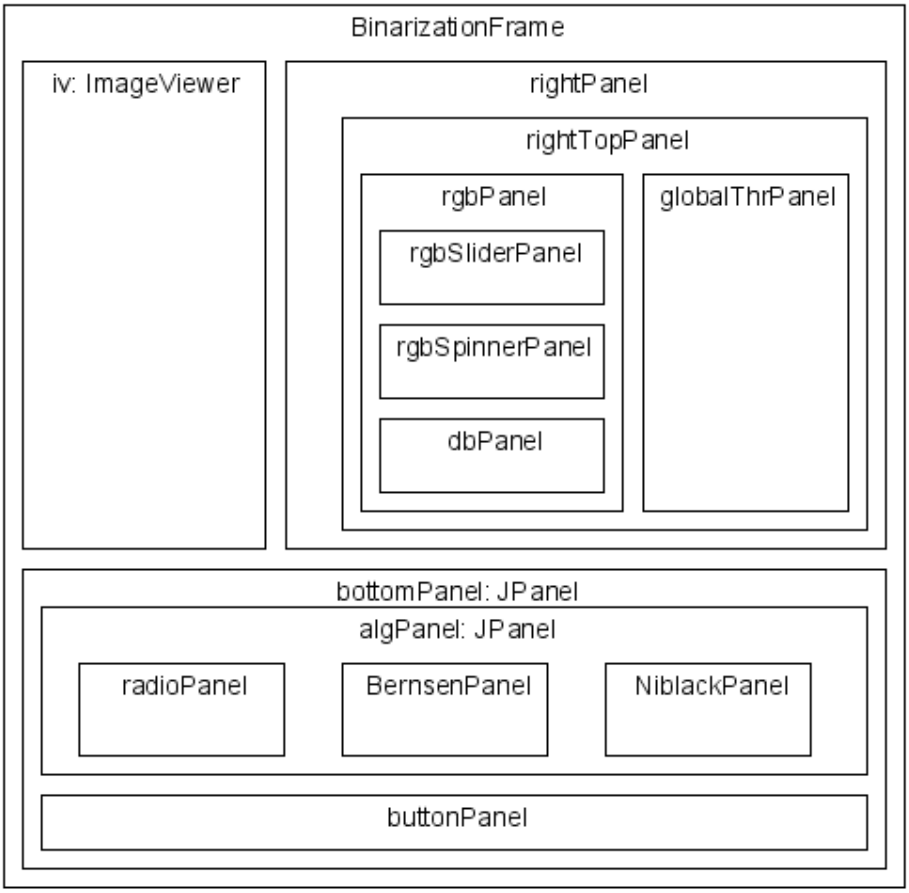
3.4.2. ResizingFrame



3.4.3. FilterFrame



3.4.4. BinarizationFrame



3.5. Undo/Redo steks

Undo/Redo steks nodrošina operāciju atcelšanu un atkārtošanu. Tas ir realizēts divu atsevišķu steku veidā: redo steks un undo steks, kuros tiek glabātas norādes uz RasterImage objektiem. Norāde uz tekošo attēlu glabājas privātajā laukā *rImage* galvenā loga klasē, kur glabājas arī steki. Galvenais logs saņem komandas "Redo", "Undo", "ffw" – pēdējais attēls, "frw" – pirmais attēls. Steka darbību apraksta pseidokods:

Komanda *Undo*	**Komanda *Redo***
redoStack.push(*rImage*) rImage=undoStack.pop	undoStack.push(*rImage*) rImage=redoStack.pop
Komanda *ffw*	**Komanda *frw***
undoStack.push(rImage) while (redoStack.size>1) undoStack.push(redoStack.pop()) rImage=redoStack.pop	redoStack.push(rImage) while (undoStack.size>1) redoStack.push(undoStack.pop()) rImage=undoStack.pop

DO tipa komandas (kuru rezultātā mainās attēls galvenajā logā)
P – jaunais attēls
redoStack.clear
if (rImage != null) undoStack.push(rImage)
rImage=P

3.6. Algoritmu projektējums

3.6.1. Binarizācija ar Bernsena metodi

1. Pārveidot attēlu pelēkuma skalā
2. Iegūt kontrasta matricu B pēc formulas $B(x,y)=\frac{B_{max}-B_{min}}{2}$, kur
   B_{max} un B_{min} ir visaugstākais un viszemākais intensitātes līmeņi kvadrātveida apgabalā ar radiusu r.
3. $B(x,y)$ – piksela ar koordinātēm (x,y) kontrasts
4. Izveidot melnbaltu attēlu U, kur
   $$U(x,y)=\begin{cases}1\ (balts),\ B(x,y)<t\\0\ (melns),\ B(x,y)\geq t\end{cases}$$
5. Atgriezt U

3.6.2. Sliekšņa (threshold) noteikšana ar Otsu metodi

Pārveidot attēlu pelēkuma skalā

Uzbūvēt intensitātes histogrammu pēc vērtībām n_i – pikselu ar intensitāti i skaita:

```
size=width*height
for i=0 to size-1 {
    intensity=intensity(pixels[i])
    h[intensity]+=1
}
```

Piezīme: šajā realizācijā netiek sastādīta normalizēta histogramma.

Noteikt minimālo un maksimālo intensitātes vērtību attēlā:

```
for i=0 to 255 {
        if (h[i]<>0) {
                minIndex=i
                break;
        }
}

for i=255 downto 0 {
        if (h[i]<>0) {
                maxIndex=i
                break;
        }
}

if (minIndex=maxIndex) return 1;    // nav iespējams sadalīt 2 klasēs
```

Inicializēt klašu sadalījumu [minIndex..minIndex] un [minIndex+1..maxIndex] un aprēķināt svērtās vidējās vērtības un vidējos intensitātes līmeņus šīm klasēm:

$$n_0 = h\left[min\ Index\right]$$

$$n_1 = size - n_0$$

$$\mu_0 = \frac{\left(min\ Index + 1\right) h\left[min\ Index\right]}{n_0}$$

$$\mu_1 = \frac{\sum_{i=min\ Index+1}^{max\ Index} \left(i+1\right) h\left[i\right]}{n_1}$$

Pārlasīt visus iespējamos sliekšņa vērtības no minIndex+1 līdz maxIndex, iteratīvi aprēķinot vidējās svērtās vērtības un vidējos intensitātes līmeņus:

```
threshold=minIndex+1
```

$$maxVariance = n_0 n_1 \left(\mu_0 - \mu_1\right)^2$$

```
for i=minIndex+2 to maxIndex {
```

$$n_0 += h\left[i-1\right]$$

$$n_1 -= h\left[i-1\right]$$

$$\mu_0 = \frac{\mu_0\left(n_0 - h[i-1]\right) + ih[i-1]}{n_0}$$

$$\mu_1 = \frac{\mu_1\left(n_1 + h[i-1]\right) - ih[i-1]}{n_1}$$

$$variance = n_0 n_1 \left(\mu_0 - \mu_1\right)^2$$

```
        if (variance>maxVariance) {
                maxVariance=variance
                threshold=i
        }
}
return threshold
```

3.6.3. Globāla attēla binarizācija, izmantojot noteikto slieksni (threshold)

```
B=new Image(width: M, height: N)
Pārveidot attēlu A pelēkuma skalā
for x: 0 to M-1 for y: 0 to N-1 {
        if (intensity(A(x, y))>=t) B(x, y)=white
        else B(x, y)=black
}
return B
```

3.6.4. Attēla izmēru palielināšana ar tuvākā kaimiņa metodi

```
B=new Image(width: M, height: N)
for x: 0 to M-1 for y: 0 to N-1 {
        for dx=k_x(x+1)-1 to k_x(x+2)-1
                for dy=k_y(y+1)-1 to k_y(y+2)-1
                        B(dx,dy)=A(x,y)
}
return B
```

3.6.5. Attēla izmēru palielināšana/samazināšana ar BLSM

```
B=new Image(width: newwidth, height: newheight)
for each RGB component {
        ky=(oldheight-1)/(newheight-1)
        kx=(oldwidth-1)/(newwidth-1)
        for i: 0 to newheight-1 {
                for j: 0 to newwidth-1 {
                        l=i*ky;
                        u=i/(L-1)*(N-1)-l;
                        c=j*kx;
                        t=j*kx-c;
                        v1=A(c, l)
                        v2=A(c+1, l)
                        v3=A(c+1, l+1)
                        v4=A(c, l+1)
                        B(j, i)=((1-t)*(1-u)*v1+t*(1-u)*v2+t*u*v3+(1-t)*u*v4);
                }
        }
}
return B
```

3.6.6. Variācijas filtrs

B=new Image(width: M, height: N)
F=copy of inputImage
arr=new int [$(2R+1)^2$]
nofiltrēt attēlu F ar vidējās vērtības filtru ar radiusu R un slieksni 0
for each RGB component {
        for i: R to N-R-1 for j: R to M-R-1 {
                sum=0
                for k: i-R to i+R for l: j-R to j+R {

$$sum += A(l;\ k)^2$$

                }

$$V = \frac{sum}{(2R+1)^2}$$

$$B(j;\ i) = V - F(j;i)^2$$

        }
}
return B

3.6.7. Attēla filtrēšana ar divvirzienu mediānas filtru

```
B=new Image(width: M, height: N)
arr=new int [2R+1]
for each RGB component {
        for i: startY to endY for j: startX to endX {
                index=0
                for k: i-r to i+r {
                        arr[index]=A(j; k)
                        index=index+1
                }
                sort(arr)
                median=arr[R]
                if (abs(median-A(j; i))>t) then B(j; i)=median
                else B(i; j) = A(i; j)
        }
        for i: startX to endX for j: startY to endY {
                index=0
                for k: i-r to i+r {
                        arr[index]=A(k; i)
                        index=index+1
                }
                sort(arr)
                median=arr[R]
                if (abs(median-A(i; j))>t) then B(i; j)=median
                else B(i; j) = A(i; j)
        }

}
return B
```

3.6.8. Attēla filtrēšana ar mediānas filtru ar kvadrātisku matricu

```
B=new Image(width: M, height: N)
arr=new int [(2R+1)^2]
for each RGB component {
        for i: startX to endX for j: startY to endY {
                index=0
                for k: i-r to i+r for l: j-r to j+r {
                        arr[index]=A(l; k)
                        index=index+1
                }
                sort(arr)
                median=arr[R]
                if (abs(median-A(i; j))>t) then B(i; j)=median
                else B(i; j) = A(i; j)
        }
}
return B
```

3.6.9. Binarizācija ar Nibleka metodi

```
B= new Image(width: M, height: N)
M=copy of A
V= copy of A
meanFilter(M, R, 0)
varianceFilter(V, R, 0)
Pārveidot attēlu A pelēkuma skalā
for i: 0 to M-1 {
        for j: 0 to N-1 {
                threshold=M(x; y)+kV(x; y)
                if (intensity(A)<threshold) then B(x; y)=white;
                else B(x; y)=black
        }
}
return B
```

3.6.10. Attēla filtrēšana ar vidējās vērtības filtru ar kvadrātisku matricu

```
B=new Image(width: M, height: N)
arr=new int [(2R+1)^2]
for each RGB component {
        for i: 0 to M-1 for j: 0 to N-1 {
                sum=0
                for k: i-r to i+r for l: j-r to j+r {
                        sum += A(l; k)
                }
                avg = sum / (2R+1)
                if (abs(avg-A(i; j))>t) then B(i; j)=avg
                else B(x; y) = A(x; y)
        }
}
return B
```

Moduļa *SphericalWave* prasību specifikācija

Dokumenta nosaukums: Programmas "AttēluApstrāde" v. 1.0 Moduļa *SphericalWave* prasību specifikācija
Dokumenta identifikators: Image.SPW.PPS.1.0
Projekta identifikators: Image
Moduļa identifikators: SPW
Autors: Jevgenijs Jonass

1. Ievads

1.1. Nolūks

Šī dokumenta nolūks ir specificēt programmas "AttēluApstrāde" moduli *SphericalWave*, lai identificētu visas lietotāju un galvenās sistēmas prasības pret moduli, kā arī radītu pamatu detalizētam projektējumam. Dokumenta mērķauditorija ir moduļa izstrādātāji, kas veiks tā projektēšanu un programmēšanu.

1.2. Darbības sfēra

Modulis *SphericalWave* paredzēts kā attēlu apstrādes programmas "AttēluApstrāde" apakšmodulis, kas atbild par teksta zīmju attēlu struktūras analīzi.

1.3. Definīcijas un saīsinājumi

Attēlu apstrāde – attēlos vai citās grafiskajās informācijas atveidošanas formās esošās informācijas analīze, parasti izmantojot signālu ciparapstrādi.
Modulis – atsevišķa identificējama programmas daļa, kuru var autonomi izveidot un izmantot, lai atvieglotu programmu sastādīšanu.
Modāls logs – logs, kas bloķē lietotāja darbu ar galveno logu līdz brīdim, kamēr lietotājs to (modālo logu) neaizvers.
SVA – sfēriska viļņa algoritms

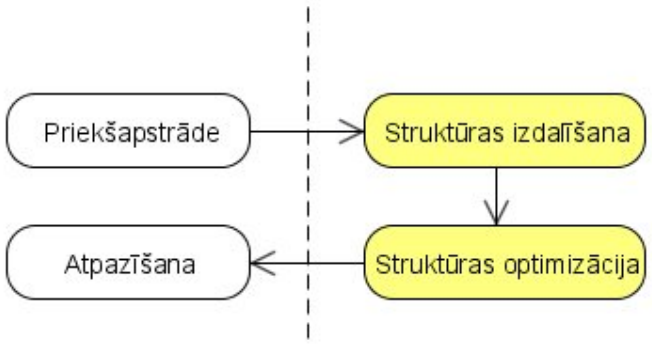
1.4. Saistība ar citiem dokumentiem

Dokuments tika izstrādāts, balstoties uz standartā LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis” aprakstītajām prasībām. Uz šī dokumenta pamata tiks izstrādāts moduļa *SphericalWave* projektējuma apraksts. Šī prasību specifikācija ir daļa no projekta "AttēluApstrāde" prasību specifikācijas: Image.PPS.1.0.

2. Vispārējs apraksts

2.1. Produkta perspektīva

Modulis paredzēts darbam ar teksta zīmju attēliem. To var izmantot kā atsevišķu teksta zīmju struktūras, tā arī teksta analīzei. Moduli var izmantot programmētāji, kā arī parastie lietotāji caur grafisko saskarni.



SPW

2.2. Moduļa funkcijas

Modulim jānodrošina:

- programmrealizācija visām funkcijām, kuras ir identificētas šajā sadaļā
- neorientēta grafa realizācija, kuras sastāvs un saskarne tiks noteikti projektēšanas laikā

Identifikators	Nosaukums
Func.SPW.1.SVA	Struktūras iegūšana ar SVA
Func.SPW.2.primopt	Šķautņu savienojumu punktu primāra optimizācija
Func.SPW.3.cne	Šķautņu galapunktu savienošana
Func.SPW.4.edgeopt	Šķautņu optimizācija
Func.SPW.5.strsav	Attēla struktūras saglabāšana failā
Func.SPW.6.strload	Attēla struktūras ielasīšana no faila
Func.SPW.7.cuttails	Lieko "astu" apcirpšana

2.3. Pieņēmumi un atkarības

Tā kā izstrādājamā programmatūra ir lielākās programmas apakšmodulis, tad tās darbībai nepieciešama pati šī programma. Modulis izmanto galvenā moduļa pakotnes un ir cieši integrēts galvenajā programmā. Modulim jābūt realizētam programmēšanas valodā *Java*, kurā tiks realizēta arī galvenā programma.

3. Funkcionālās prasības

3.1. Struktūras iegūšana ar SVA

Identifikators	Func.SPW.1.SVA
Nosaukums	Struktūras iegūšana ar SVA

Mērķis

Iegūt teksta zīmju attēla struktūru, imantojot sfēriskā viļņa algoritmu.

Ievaddati

Binārs teksta zīmju attēls, kur balti pikseli atbilst fonam, melni – teksta zīmēm.

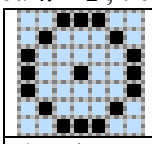
$n \in \mathbb{Z}_{>0}$ – minimālais viļņa ģenerācijas izmērs pikselos

$k \in \{1; 2\}$ – izvēlētā metode nākamās viļņa ģenerācijas iegūšanai: 16-saistība vai 4/8-saistība

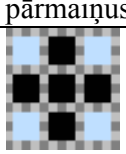
Apstrāde

Funkcija izveido jaunu tukšu grafu.
Tiek pēc kārtas apstrādāti visi melni pikseli, no katra palaižot vilni.
Funkcija seko katras viļņa ģenerācijas vidējam punktam, tādā veidā iegūstot slīpu līniju.
Kad vilnis sadalās vairākos viļņos vai nonāk strupceļā, iegūtais ceļš tiek saglabāts grafā.

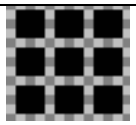
Ja $k = 1$, viļņa ģēnerācijas iegūšanai tiek izmantoti 16 blakus esošie pikseli (Att. 1).
Ja $k = 2$, tiek pārmaiņus izmantota 4-saistīta (Att. 2) un 8-saistīta (Att. 3) viļņa izplatīšanās.



Att. 1



Att. 2



Att. 3

Izvaddati

Neorientēts grafs, kas reprezentē teksta zīmju struktūru. Vispārīgā gadījumā grafs nav saistīts.

3.2. Šķautņu savienojumu punktu primāra optimizācija

Identifikators	Func.SPW.2.primopt
Nosaukums	Šķautņu savienojumu punktu primāra optimizācija

Mērķis

Funkcija veic struktūras optimizāciju, iztaisnojot nogriežņus, kas grafā ir reprezentēti ar divām šķautnēm, kas iziet no vienas virsotnes un ir gandrīz paralēlas, bet pretēji vērstas.

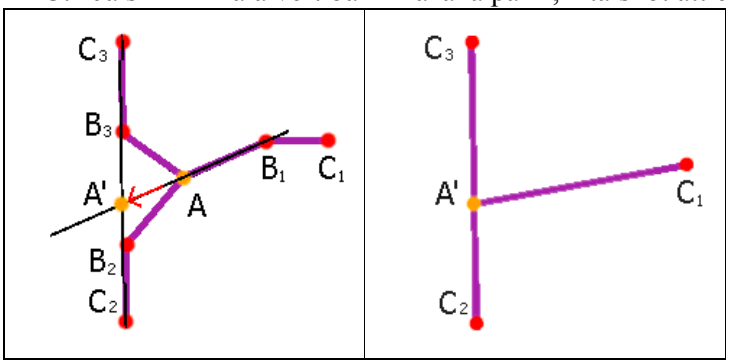
Ievaddati

1. Grafs, kas reprezentē teksta zīmju struktūru
2. Sliekšņa kosinusa vērtība $k \in [-1;1]$

Apstrāde

Meklēt virsotnes ar pakāpi 3. Katrai tādai virsotnei:

1. Izrēķināt kosinusus lenķiem starp vektoriem no dotās virsotnes uz katru no 3 tās kaimiņvirsotnēm (lenķis ir robežās no 0 līdz 180 grādiem)
2. Izrēķināt minimālo no 3 iegūtajiem kosinusiem
3. Ja šī minimālā vērtība ir mazāka par k, iztaisnot attiecīgo nogriezni:



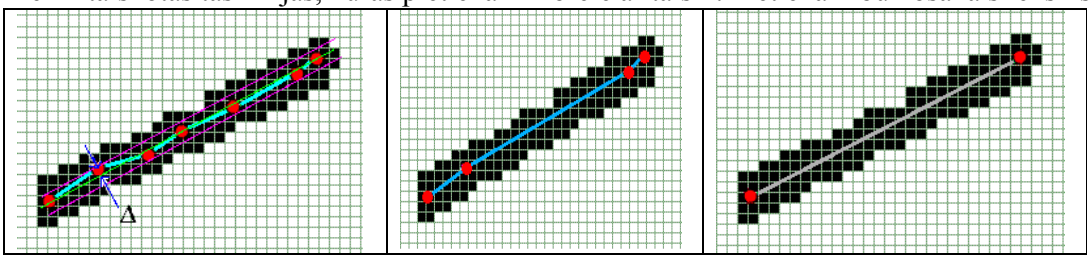
Izvaddati

Optimizēts grafs.

3.3. Šķautņu galapunktu savienošana

Identifikators	Func.SPW.3.cne
Nosaukums	Šķautņu galapunktu savienošana
Mērķis	
SVA algoritmā netiek savienotas virsotnes, kad vilnis nonāk "strupceļā". Šī funkcija savieno šādas virsotnes.	
Ievaddati	
Grafs, kas reprezentē teksta zīmju attēla struktūru. Attāluma slieksnis $d \in \mathbb{R}_{\geq 0}$	
Apstrāde	
Tiek savienoti tās virsotnes ar pakāpi 1, attālums starp kurām ir mazāks par d: 	
Izvaddati	
Optimizēts grafs.	

3.4. Šķautņu optimizācija

Identifikators	Func.SPW.4.edgeopt
Nosaukums	Šķautņu optimizācija
Mērķis	
SVA algoritma darbības rezultātā iegūtās līnijas ne vienmēr ir gludas. Šī funkcija iztaisno šādas līnijas.	
Ievaddati	
Grafs, kas reprezentē teksta zīmju attēla struktūru. Korelācijas slieksnis $k \in \mathbb{R}_{\geq 0}$	
Apstrāde	
Tiek iztaisnotas tās līnijas, kuras pietiekami korelē ar taisni. Pietiekamību nosaka slieksnis k. 	
Izvaddati	
Optimizēts grafs.	

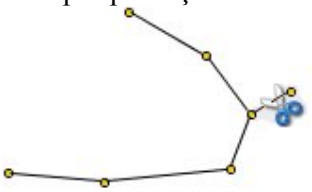
3.5. Attēla struktūras saglabāšana failā

Identifikators	Func.SPW.5.strsav
Nosaukums	Attēla struktūras saglabāšana failā
Mērķis	
Saglabāt attēla struktūru (grafu) failā.	
Ievaddati	
1. Grafs, kas reprezentē teksta zīmju attēla struktūru. 2. Java standartklases *File* tipa objekts	
Apstrāde	
Grafa dati tiek saglabāti failā.	
Izvaddati	
Nav.	

3.6. Attēla struktūras ielasīšana no faila

Identifikators	Func.SPW.6.strload
Nosaukums	Attēla struktūras ielasīšana no faila
Mērķis	
Ielādēt attēla struktūru (grafu) no faila.	
Ievaddati	
Java standartklases *File* tipa objekts	
Apstrāde	
Attēla dati tiek nolasīti no faila.	
Izvaddati	
Grafs, kas reprezentē teksta zīmju attēla struktūru.	

3.7. Lieko "astu" apcirpšana

Identifikators	Func.SPW.7.cuttails
Nosaukums	Lieko "astu" apcirpšana
Mērķis	
Apcirpt liekas grafa daļas.	
Ievaddati	
Grafs, kas reprezentē teksta zīmju attēla struktūru. Šķautņu secības garuma slieksnis $l \in \mathbb{R}_{\geq 0}$	
Apstrāde	
Tiek apcirptas šķautnes vai šķautņu secības, kuras ir īsākas par l: 	
Izvaddati	
Optimizēts grafs.	

4. Grafiskā saskarne

Modulis ir galvenokārt paredzēts izmantošanai caur programmsaskarni, bet tam ir arī grafiskā lietotāja saskarne. Grafiskajai saskarnei jānodrošina iespēja izplildīt visas funkcijas, kuras ir aprakstītas 3. nodaļā.

4.1. Attēla struktūras analīzes logs

Nemodāls, pārvietojams maināma logs ar maināmu izmēru, kas ļauj pielietot izvēlētos algoritmus, vizualizēt to darbības rezultātus, ievadīt nepieciešamos parametrus, kā arī saglabāt rezultātus. Pēc loga atvēršanas tajā parādās attēla kopija no galvenā programmas loga. Pēc loga aizvēršanas visas veiktās izmaiņas tiek pazaudētas.

Komponents	Funkcija
panele	parāda ielādēto attēlu un uz tā uzzīmētu struktūru
poga *SVA*	Func.SPW.1.SVA
poga *Primary opt.*	Func.SPW.2.primopt
poga *Connect edges*	Func.SPW.3.cne
poga *Edge opt.*	Func.SPW.4.edgeopt
poga *Cut tails*	Func.SPW.7.cuttails
5 teksta lauki	ļauj ievadīt visu izsaucāmo funkciju parametrus.
nolaižamais sarakstlodziņš	ļauj izvēlēties funkcijas Func.SPW.1.SVA režīmu: 16-saistība vai 4/8-saistība
izvēlne *Graph* ar punktiem *Save* un *Load*	kalpo attēla struktūras (grafa) saglabāšanai/ielasīšanai no faila

4.2. Faila atvēršanas un saglabāšanas logi

Šie logi paredzēti grafa ielādēšanai vai saglabāšani failā, izmantojot lietotāja saskarni. Tie paredz sekojošus kļūdu paziņojumus, kas rodas faila rakstīšanas/lasīšanas kļūdas gadījumā, kā arī faila formāta kļūdas gadījumā:

- "Error occurred while reading from file."
- "Error occurred while writing to file."

5. Nefunkcionālās prasības

5.1. Veiktspējas prasības

Netiek izvirzītas nekādas prasības pret ātrdarbību. Tomēr tiek izvirzītas prasības pret datu apjomiem: modulim jāspēj apstrādāt attēlus ar izmēriem līdz 1280x1280 un grafus ar virsotņu skaitu līdz 1000. Modulis neļauj ielādēt no faila lielākus grafus.

5.2. Faila formāts

Formāts, kādā grafs tiek glabāts failā, tiks precizēts moduļa PPA.

Moduļa *SphericalWave* projektējuma apraksts

Dokumenta nosaukums: Programmas „AttēluApstrāde” v. 1.0 Moduļa *SphericalWave* Programmatūras projektējuma apraksts
Dokumenta identifikators: Image.SPW.PPA.1.0
Projekta identifikators: Image
Moduļa identifikators: SPW
Autors: Jevgenijs Jonass

1. Ievads

1.1. Nolūks

Šī dokumenta nolūks ir aprakstīt programmas "AttēluApstrāde" moduļa *SphericalWave* projektējumu, atspoguļojot prasības pret moduli. Dokumentā netiks aprakstītas klašu metodes, jo tas tiks izdarīts ar *Javadoc* dokumentēšanas rīku. Dokumenta mērķauditorija ir moduļa izstrādātāji.

1.2. Darbības sfēra

Modulis *SphericalWave* paredzēts kā attēlu apstrādes programmas "AttēluApstrāde" apakšmodulis, kas atbild par teksta zīmju attēlu struktūras analīzi.

1.3. Definīcijas un saīsinājumi

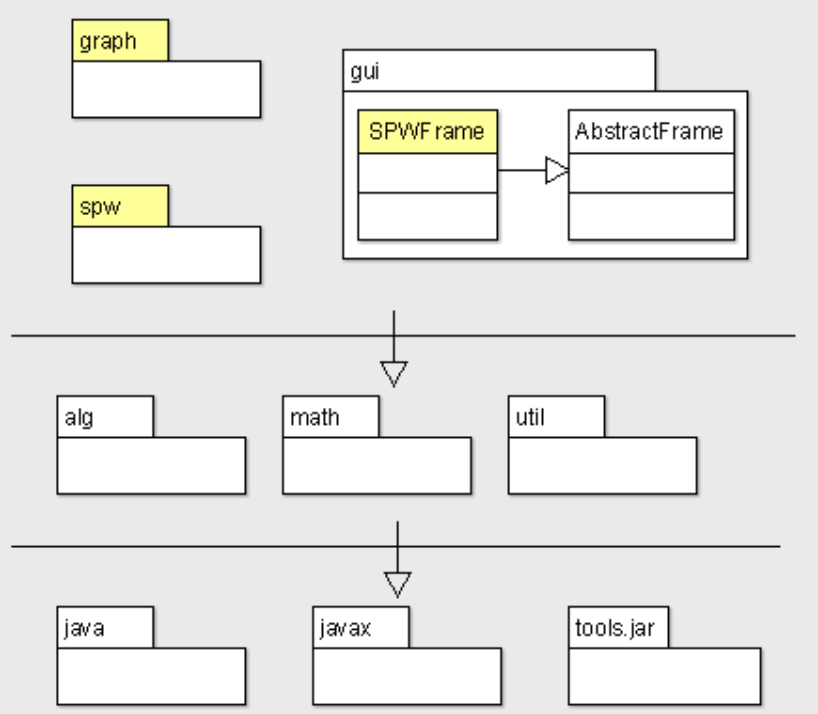
Attēlu apstrāde – attēlos vai citās grafiskajās informācijas atveidošanas formās esošās informācijas analīze, parasti izmantojot signālu ciparapstrādi.
Modulis – atsevišķa identificējama programmas daļa, kuru var autonomi izveidot un izmantot, lai atvieglotu programmu sastādīšanu.
Modāls logs – logs, kas bloķē lietotāja darbu ar galveno logu līdz brīdim, kamēr lietotājs to (modālo logu) neaizvers.
Pakotne (pakete) – Java klašu kopums
Grafs – matemātiska sistēma (V, E), kur V ir virsotņu kopa, bet E – apakškopa noVxV.
SVA – sfēriska viļņa algoritms

1.4. Saistība ar citiem dokumentiem

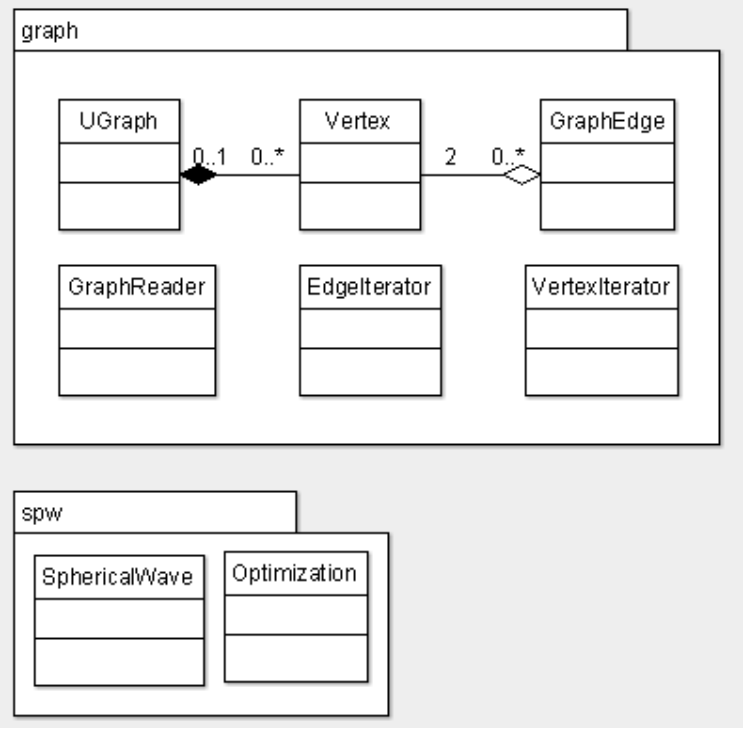
Dokuments tika izstrādāts, balstoties uz standartā LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai” aprakstītajām prasībām. Šis programmatūras projektējuma apraksts ir daļa no programmas "AttēluApstrāde" projektējuma apraksta: Image.PPA.1.0. Šī dokumeta sastāvdaļa ir projekta "AttēluApstrāde" moduļa *SphericalWave* *Javadoc* dokumentācija.

2. Dekompozīcijas apraksts

2.1. Pakotņu diagramma



2.2. Klašu diagramma



2.3. Klašu apraksts

Pakotne *graph*

Klase	Apraksts
UGraph	klase reprezentē neorientētu grafu, kura virsotnes atbilst attēla pikseliem
Vertex	klase reprezentē grafa virsotni, kas raksturojas ar koordinātām
GraphEdge	palīgklase, kas inkapsulē 2 grafa virsotnes
GraphReader	klase satur metodes grafa saglabāšanai/ielasīšanai no faila
EdgeIterator	klase ļauj iterēt caur visām grafa šķautnēm
VertexIterator	klase ļauj iterēt caur grafa virsotnēm

Pakotne *spw*

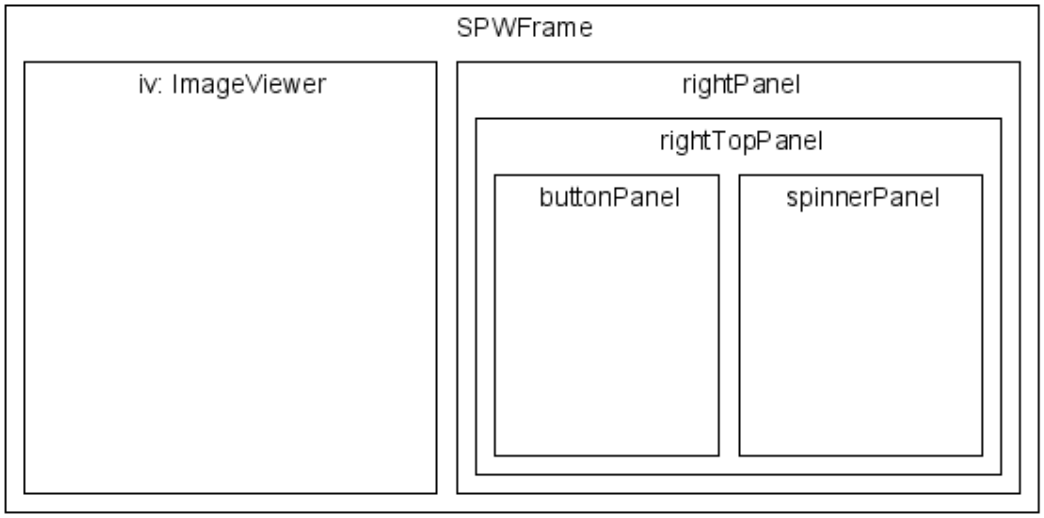
Klase	Apraksts
SphericalWave	klase satur metodes attēla struktūras iegūšanai ar SVA algoritmu
Optimization	klase satur metodes attēla struktūras optimizācijai

Pakotne *gui*

Klase	Apraksts
SPWFrame	Reprezentē moduļa grafisko logu.
...	Citu moduļu grafiskās komponentes, kas neattiecas uz doto moduli.

3. Projektējuma apraksts

3.1. Grafiskā loga projektējums



3.2. Faila formāts

Grafs tiek glabāts teksta failā sekojošā formātā: pirmā rinda satur virsotņu skaitu N un šķautņu skaitu M. Nākamās N rindas satur virsotņu koordinātas, pēdējās M rindas satur šķautņu aprakstu, kas sastāv no pirmās un otrās virsotnes koordinātēm. Visi skaitļi vienā rindā ir atdalīti ar tukšuma zīmēm.

$N\ M$

$V_{1x}\ V_{1y}$

$V_{2x}\ V_{2y}$

...

$V_{Nx}\ V_{Ny}$

$P_{1x}\ P_{1y}\ Q_{1x}\ Q_{1y}$

$P_{2x}\ P_{2y}\ Q_{2x}\ Q_{2y}$

...

$P_{Mx}\ P_{My}\ Q_{Mx}\ Q_{My}$

3.3. EdgeIterator klase

```
class EdgeIterator {
        edge=[null, null]
        iterated = ∅
        neighbours=[]
        K = ∅
        konstruktors EdgeIterator(g) {
                K ← {virsotņu kopa}
                if ( K = ∅ ) return
                V ← K.remove
                while (V.degree=0) {
                        if ( K = ∅ ) return
                        V ← K.remove
                }
                neighbours=V.neighbours
                edge=[V; neighbours.next]
        }

        next {
                do {
                        while (neighbours.hasNext=false) {
                                iterated=iterated ∪ {edge[0]}
                                if ( K = ∅ ) {
                                        tmp=edge
                                        edge=[null, null]
                                        return tmp
                                }
                                edge[0]=K.remove
                                neighbours=edge[0].neighbours
                        }
                        edge[1]=neighbours.next
                }
                while ( edge[0] ∈ iterated )
                return edge
```

```
        }

        hasNext {
                if edge=[null, null] return false
                else return true
        }
}
```

3.4. Algoritmu projektējums

3.4.1. Struktūras iegūšana ar SVA

1. Inicializēt tukšu grafu g;
2. Izveidot tukšu steku, kurā glabāsies viļņa ģenerācijas;
3. Ievietot tajā jebkuru attēla punktu kā viļņa ģenerāciju;
4. Kāmēr steks nav tukšs:
    a. Izņemt viļņa ģenerāciju G no steka;
    b. Aprēķināt kopas G vidējo svērto punktu C
    c. Sadalīt viļņa ģenerāciju apakškopās, balstoties uz 8-saistības
    d. Aprēķināt katras apakškopas S_i vidējo svērto punktu V_i
    e. Pievienot punktu V_i grafam un savienot to ar punktu C
    f. Ielikt katru apakškopu S_i stekā kā viļņa ģenerāciju

3.4.2. Šķautņu savienojumu punktu primāra optimizācija

$K \leftarrow \{virsotnes\ ar\ pakāpi\ 3\}$

```
while K ≠ ∅ {
        V ← K.remove
        (P,Q,R) ← V.neighbours
        cos12=cos(VP, VQ)
        cos13=cos(VP, VR)
        cos23=cos(VQ, VR)
        minCos=min(cos12, cos13, cos23)
        if (minCos<k) {
                if (minCos=cos12) {
                        (x,y) ← taišņu_krustpunkts(RV,QP)
                        V.delete
                        W = g.add(x,y)
                        W.addneighbours(P, Q, R)
                }
                else if (minCos=cos13) {
                        (x,y) ← taišņu_krustpunkts(QV,PR)
                        V.delete
                        W = g.add(x,y)
                        W.addneighbours(P, Q, R)
                }
                else if (minCos=cos23) {
                        (x,y) ← taišņu_krustpunkts(PV,QR)
                        V.delete
```

```
                    W = g.add(x, y)
                    W.addneighbours(P, Q, R)
                }
        }
}
return g
```

3.4.3. Šķautņu galapunktu savienošana

$K \leftarrow \{virsotnes\ ar\ pakāpi\ 1\}$

```
for (V: K) for (W: K) {
        if (distance(W, V)<k) W.addneighbour(V)
}
return g
```

3.4.4. Šķautņu optimizācija

Uzkonstruēt visu grafa šķautņu kopu K. Kāmēr šī kopa nav tukša, pildīt punktus 1–7

1. Paņemt jebkuru šķautni E no kopas K
2. Inicializēt tukšu virsotņu sarakstu S
3. Pievienot sarakstam S abas šķautnes E virsotnes jebkurā secībā
4. Pildīt punktus 4.a – 4.g , kāmēr sarakstam S tiek pievienota vismaz viena virsotne (added1 vai added2 ir true)
    a. Inicializēt mainīgos added1:=false, added2:=false
    b. Paņemt pēdējo virsotni V no saraksta S
    c. Ja virsotnes V pakāpe vienāda ar 2 un eksistē tās kaimiņš W, kas nav sarakstā S (tāds var būt tikai 1 no 2 kaimiņiem) un šķautne (W; V) pieder K:
        i. Pievienot virsotni W saraksta S beigās
        ii. izdzēst šķautni (W; V) no kopas K
        iii. added1:=true
    d. Ja added1==true:
        i. izrēķināt dispersiju iegūtam virsotņu sarakstam
        ii. ja iegūtais koeficients lielāks par k, izdzēst pēdējo virsotni (W) no saraksta S un piešķirt added1:=false
    e. Paņemt pirmo virsotni Q no saraksta S
    f. Ja virsotnes Q pakāpe vienāda ar 2 un eksistē tās kaimiņš R, kas nav sarakstā S (tāds var būt tikai 1 no 2 kaimiņiem) un šķautne (Q; R) pieder K:
        i. Pievienot virsotni R saraksta S sākumā
        ii. izdzēst šķautni (Q; R) no kopas K
        iii. added2:=true
    g. Ja added2==true:
        i. izrēķināt dispersiju iegūtam virsotņu sarakstam
        ii. ja iegūtais koeficients lielāks par k, izdzēst pirmo virsotni (R) no saraksta S un piešķirt added2:=false
5. Izdzēst no grafa visas saraksta S šķaunes, izņemot pirmo un pēdējo.
6. Pievienot grafam šķautni starp pirmo un pēdējo virsotni sarakstā S (ja tāda neeksistēja)
7. Izdzēst šo šķautni no kopas K (ja šī kopa to saturēja)

return g

3.4.5. Lieko "astu" apcirpšana

```
izdzēst virsotnes  ar pakāpi 0
K ← {virsotnes ar pakāpi 1}
Iterator it=K.iterator
while (it.hasNext) {
        V ← it.next
        List L=[V]
        if (V.degree=0) continue
        W=V.neighbour
        double len=distance(V, W)
        while (len<maxLen) {
                L.add(W)
                if (W.degree ≠ 2) break
                len += distance(W, W.nextNeighbour)
                W=W.nextNeighbour
        }
        L.deleteLast
        g.deleteVertices(L)
}
izdzēst virsotnes  ar pakāpi 0
return g
```

Testēšanas plāns

Dokumenta identifikators: Image.UT.TP.1.0
Projekta identifikators: Image
Autors: Jevgenijs Jonass

1. Ievads

Šis testēšanas plāns apraksta programmas "AttēluApstrāde" moduļu testēšanu. Šajā plānā paredzēts notestēt 2 moduļus: *Main* un *SphericalWave*.

2. Testējamie vienumi

Pakotne	Klase	Vai tiks testēts	Statuss
alg	Binarization	☒ Jā ☐ Nē	☒ Notestēts
	Bresenham	☐ Jā ☒ Nē	☐ Notestēts
	MeanFilter	☐ Jā ☒ Nē	☐ Notestēts
	MedianFilter	☒ Jā ☐ Nē	☒ Notestēts
	SortedArray	☒ Jā ☐ Nē	☒ Notestēts
	Spline	☐ Jā ☒ Nē	☐ Notestēts
gui	–	☐ Jā ☒ Nē	☐ Notestēts
math	Line	☒ Jā ☐ Nē	☒ Notestēts
	CoordVector	☒ Jā ☐ Nē	☒ Notestēts
graph	EdgeIterator	☒ Jā ☐ Nē	☒ Notestēts
	GraphEdge	☐ Jā ☒ Nē	☐ Notestēts
	GraphReader	☒ Jā ☐ Nē	☒ Notestēts
	UGraph	☒ Jā ☐ Nē	☒ Notestēts
	Vertex	☒ Jā ☐ Nē	☒ Notestēts
	VertexIterator	☒ Jā ☐ Nē	☒ Notestēts
util	–	☐ Jā ☒ Nē	☐ Notestēts
spw	SphericalWave	☐ Jā ☒ Nē	☐ Notestēts
	Optimization	☒ Jā ☐ Nē	☒ Notestēts

3. Testējamās raksturiezīmes

1) Jāsalīdzina, vai funkcionalitāte atbilst projektējuma aprakstam un metodes specifikācijai.
2) Tā kā laiks ir ierobežots, netiks testētas visas metodes, kā arī metožu darbības korektums netiks pakļauts visām būtiski atšķirīgajām ievaddatu kombinācijām, bet tikai dažām no tām.
3) Jānotestē metodes uz robežvērtībām.

4. Netestējamās raksturiezīmes

1) Metodes tiks testētas, balstoties uz pieeju *Design By Contract*: tiks pārbaudīts, vai metodes izpilda izvirzītās prasības (jeb pēcnosacījumus), ja izpildās priekšnosacījumi. Metodes netiks testētas uz ieejas datiem, kuri nav paredzēti metodes specifikācijā.
2) Šī testēšanas plāna ietvaros netiks testēta ātrdarbība un slodze.

5. Pieeja

Testēšanas līmenis: moduļu testēšana. Testēšana notiks pēc baltās kastes metodes. Testi tiks sastādīti balstoties uz moduļu PPS, PPA un programmas kodu. Tiks izmantots Java modulis *JUnit*. Tiks izveidota direktoriju hierarhija ar sakni "<Projekta mape>/test/", kas atbilst izejas koda pakotņu hierarhijai. Katrai testējamai klasei <*Class*> tiks izveidota klase <*ClassTest*>.

6. Testējamā vienuma novērtēšanas kritēriji

Testējamā klase ir izturējusi testēšanu, ja visi testi ir veiksmīgi, ko noteiks *JUnit* parādītie rezultāti.

7. Atlikšanas kritēriji un atsākšanas prasības

Ja testēšanas rezultātā vismaz viens tests atklājis kļūdu, klase tiek nodota atkļūdošanai. Laikā, kad tiek labota kļūdainā klase, testētājs var sākt pārbaudīt nākamo klasi un atgriezties pie kļūdainās, kad tā būs izlabota.

8. Testēšanas nodevumi

Pabeidzot testēšanu ir jānodod šādi dokumenti:

- Image.UT.TP.1.0 – Testēšanas plāns (šis dokuments)
- Image.UT.TZ.1.0 – Testēšanas žurnāls
- Image.UT.TS.Main.1.0 un Image.UT.TS.SPW.1.0 – Moduļu testpiemēru specifikācijas

9. Testēšanas uzdevumi

1) Jāpārbauda, vai ir programmas koda pēdēja versija ir nokopēta no repozitorija uz testētāja datora.
2) Jāpārbauda, vai Java platforma darbojas pareizi.
3) Jāpalaiž NetBeans.
4) Jāsagatavo testēšanas dokumenti, piemēram, jāatver jauns žurnāls u.tml.

10. Vides vajadzības

1) Testētāji jānodrošina ar datoriem, uz kuriem būtu uzinstalēts JDK un JRE, NetBeans ar JUnit moduli.
2) Uz testētāju datoriem jābūt testējamās programmas izejas kodam.
3) Jānodrošina piekļuve SVN repozitorijam.

11. Atsauces uz citiem dokumentiem

- Programmatūras prasību specifikācijas: Image.PPS.1.0, Image.SPW.PPS.1.0
- Programmatūras projektējuma apraksti: Image.PPA.1.0, Image.SPW.PPA.1.0
- Projekta "AttēluApstrāde" *Javadoc* dokumentācija

Moduļa *Main* testpiemēru specifikācija

Dokumenta identifikators: Image.UT.TS.Main.1.0
Projekta identifikators: Image
Moduļa identifikators: Main
Autors: Jevgenijs Jonass

1. Klase MedianFilter

1.1. Testpiemēru kopa 1

Testējamā metode: `public static void medianFilter`

Metode tiks testēta uz pelēkiem attēliem ar filtrēšanas slieksni 0.

Testēšanas procedūra

Katrā testpiemērā tiek sagatavoti 2 attēli – oriģinālais un sagaidāmais. Oriģinālais attēls tiek nolasīts no faila, konvertēts uz formātu (klasi) *RasterImage*. Pēc tam tas tiek nofiltrēts ar testējamo metodi, iegūtie attēli tiek salīdzināti ar metodi `RasterImage.equals`.

Testpiemēru sagatavošana

Testpiemēru sagatavošanai tiek izmantota mediānas filtra realizācija projekta *BookScanLib* ietvaros (http://djvu-soft.narod.ru/bookscanlib/009.htm).
Testpiemēru sagatavošanai būs vajadzīgi faili:

- Mediānas filtra realizācija: despec1.exe, runme.bat (http://djvu-soft.narod.ru/bookscanlib/despec1.rar)
- FreeImage.dll (http://djvu-soft.narod.ru/bookscanlib/freeimage_dll_v392.rar)

Tā kā *BookScanLib* atbalsta tikai *grayscale tiff* formātu, bet programma "AttēluApstrāde" to neatbalsta, oriģinālais attēls jākonvertē uz *grayscale bmp* ar programmas "AttēluApstrāde" palīdzību, pēc tam ar *Paint* palīdzību uz formātu *grayscale tiff*, kurā to filtrē ar *BookScanLib*. Iegūtais rezultāts jākonvertē uz 24 bitu *bmp* formātu, pēc tam uz *grayscale bmp*. Sagatavošanas rezultātā jābūt:

- ieejas attēlam formātā *grayscale bmp*
- ar *BookScanLib* filtrētam attēlam formātā *grayscale bmp*

Nepieciešamie faili

Visi attēli atrodas repozitorijā, mapē AtteluApstrade/testdata/MedianFilterTest.

1.1.1. Testpiemērs 1

Testpiemēra ID: s1t1
Apraksts: metode tiek testēta ar radiusu 1.
Oriģinālais attēls: s1t1.orig.gray.bmp
Sagaidāmais attēls: s1t1.exp.gray.bmp

1.1.2. Testpiemērs 2

Testpiemēra ID: s1t2
Apraksts: metode tiek testēta ar radiusu 5.
Oriģinālais attēls: s1t2.orig.gray.bmp
Sagaidāmais attēls: s1t2.exp.gray.bmp

1.1.3. Testpiemērs 3

Testpiemēra ID: s1t3
Apraksts: metode tiek testēta ar radiusu 20.
Oriģinālais attēls: s1t3.orig.gray.bmp
Sagaidāmais attēls: s1t3.exp.gray.bmp

2. Klase SortedArray

Katrā testpiemērā klases *SortedArray* objekts tiks novests dotajā stāvoklī, inicializējot to ar metodi `init`, kurai tiks padots iepriekš izveidotais buferis. Inicializācija notiks tikai izmantojot parametru `shift=0`. Pēc tam tiks izsaukta metode `replace` un tiks pārbaudīti lauki `arr` un `q`. Šo lauku vērtības tiks salīdzinātas ar šajā specifikācijā noteiktajām vērtībām. Masīvs `arr` tiks salīdzināts ar metodes *arrayEquals* palīdzību, bet rinda `q` tiks salīdzināta, izņemot pēc kārtas visus elementus un salīdzinot šos elementus, kā arī salīdzinot rindu izmērus. Tiks pārbaudīts arī, vai metode `average()` atgriež pareizo vērtību.

2.1. Testpiemēru kopa 1

<table>
<tr><th>Testa ID</th><th>Klases stāvoklis pirms ievades</th><th>Ievade</th><th>Klases lauku vērtības</th></tr>
<tr><td>s1t1</td><td rowspan="4">arr:
2 | 8 | ~~13~~ | 14 | 18 | 29 | 31
q:
13 | 18 | 8 | 29 | 31 | 2 | 14</td><td>replace(14)</td><td>arr:
2 | 8 | 14 | 14 | 18 | 29 | 31
q:
18 | 8 | 29 | 31 | 2 | 14 | 14
average()=16,57142857142857142</td></tr>
<tr><td>s1t2</td><td>replace(13)</td><td>arr:
2 | 8 | 13 | 14 | 18 | 29 | 31
q:
18 | 8 | 29 | 31 | 2 | 14 | 13
average()=16,42857142857142857</td></tr>
<tr><td>s1t3</td><td>replace(8)</td><td>arr:
2 | 8 | 8 | 14 | 18 | 29 | 31
q:
18 | 8 | 29 | 31 | 2 | 14 | 8
average()=15,71428571428571428</td></tr>
<tr><td>s1t4</td><td>replace(16)</td><td>arr:
2 | 8 | 14 | 16 | 18 | 29 | 31
q:
18 | 8 | 29 | 31 | 2 | 14 | 16
average()=16,85714285714285714</td></tr>
</table>

s1t5	arr: 2 2 2 8 ~~15~~ 15 q: 15 8 2 15 2 2	replace(8)	arr: 2 2 2 8 8 15 q: 8 2 15 2 2 8 average()=6,166666666666666666
s1t6		replace(15)	arr: 2 2 2 8 15 15 q: 8 2 15 2 2 15 average()=7,333333333333333333
s1t7		replace(2)	arr: 2 2 2 2 8 15 q: 8 2 15 2 2 2 average()=5,16666666666666666
s1t8		replace(9)	arr: 2 2 2 8 9 15 q: 8 2 15 2 2 9 average()=6,33333333333333333

2.2. Testpiemēru kopa 2

s2t1	arr: ~~5~~ q: 5	replace(5)	arr: 5 q: 5 average()=5
s2t2		replace(7)	arr: 7 q: 7 average()=7
s2t3		replace(2)	arr: 2 q: 2 average()=2
s2t4	arr: 0 0 ~~0~~ 2 2 3 8 q: 0 2 3 2 8 0 0	replace(16)	arr: 0 0 2 2 3 8 16 q: 2 3 2 8 0 0 16 average()=4,428571428571428

s2t5		replace(30)	arr: 5 | 8 | 15 | 30 q: 8 | 5 | 15 | 30 average()=14,5
s2t6		replace(18)	arr: 5 | 8 | 15 | 18 q: 8 | 5 | 15 | 18 average()=11,5
s2t7	arr: 5 | 8 | 15 | ~~20~~	replace(3)	arr: 3 | 5 | 8 | 15 q: 8 | 5 | 15 | 3 average()=7,75
s2t8	q: 20 | 8 | 5 | 15	replace(13)	arr: 5 | 8 | 13 | 15 q: 8 | 5 | 15 | 13 average()=10,25
s2t9		replace(7)	arr: 5 | 7 | 8 | 15 q: 8 | 5 | 15 | 7 average()=8,75
s2t10		replace(100) replace(0) replace(-30)	arr: -30 | 0 | 15 | 100 q: 15 | 100 | 0 | -30 average()=21,25

3. Klase CoordVector

3.1. Testpiemēru kopa 1

3.1.1. Testpiemērs 1 – konstruktors CoordVector(double... args)

Testpiemēra ID: s1t1
Apraksts: tiek izveidots jauns *CoordVector* objekts ar konstruktoru, kuram padod ka parametrus skaitļus 3 un 4.
Sagaidāmais rezultāts: lauks `vector` vienāds ar (3, 4), lauks squares vienāds ar (9, 16)

3.1.2. Testpiemērs 2 – konstruktors CoordVector(int dim)

Testpiemēra ID: s1t2
Apraksts: tiek izveidots jauns *CoordVector* objekts ar konstruktoru, kuram padod ka parametru skaitli 3
Sagaidāmais rezultāts: lauks `vector` vienāds ar (0, 0, 0), lauks squares vienāds ar (0, 0, 0)

3.1.3. Testpiemērs 3 – metode setElement

Testpiemēra ID: s1t3
Apraksts: tiek izveidots jauns 3-dimensiju *CoordVector*, kas satur skaitļus (0, 0, 0); vektoram tiek pielietota metode `setElement(2, 10.5)`
Sagaidāmais rezultāts: metode `getElement(2)` atgriež 10.5; lauka squares[2] vērtība ir 110,25

3.1.4. Testpiemērs 4 – konstruktors CoordVector(double... args)

Testpiemēra ID: s1t4
Apraksts: tiek izveidots jauns *CoordVector* objekts ar konstruktoru, kuram padod ka parametrus skaitļus 4, 0.5, -8, 5, 2, 0.
Sagaidāmais rezultāts: lauks `vector` vienāds ar (4, 0.5, -8, 5, 2, 0), lauks squares vienāds ar (16, 0.25, 64, 25, 4, 0)

3.1.5. Testpiemērs 5 – metode scalarProduct

Testpiemēra ID: s1t5
Apraksts: tiek izveidoti 2 vektori (4, 5) un (-2, 3); tiek izsaukta metode `scalarProduct`, kurai padod kā parametrus abus vektorus
Sagaidāmais rezultāts: metode atgriež skaitli 7

4. Klase Line

4.1. Testpiemēru kopa 1

4.1.1. Testpiemērs 1 – metode angleBetweenLinesCos(double x1, double y1, double x2, double y2, double x3, double y3)

Testpiemēra ID: s1t1
Apraksts: tiek izsaukta metode `angleBetweenLinesCos`, kurai padod kā parametrus 1, 1, 2, 2, 3, 3
Sagaidāmais rezultāts: metode atgriež skaitli 1

4.1.2. Testpiemērs 2 – metode angleBetweenVectorsCos(double x1, double y1, double x2, double y2, double x3, double y3)

Testpiemēra ID: s1t2
Apraksts: tiek izsaukta metode `angleBetweenVectorCos`, kurai padod kā parametrus 1, 1, 2, 2, 1, 1
Sagaidāmais rezultāts: metode atgriež skaitli 0

4.1.3. Testpiemērs 3 – metode distanceFromLineToPoint(double x1, double y1, double Ax, double Ay, double Bx, double By)

Testpiemēra ID: s1t3
Apraksts: tiek izsaukta metode `distanceFromLineToPoint`, kurai padod kā parametrus 1, 3, 0, 0, 4, 4
Sagaidāmais rezultāts: metode atgriež skaitli 1,41421356237310

4.1.4. Testpiemērs 4 – metode distanceFromLineToPoint(double x1, double y1, double Ax, double Ay, double Bx, double By)

Testpiemēra ID: s1t4
Apraksts: tiek izsaukta metode `distanceFromLineToPoint`, kurai padod kā parametrus 1, -1, -1, -2, 4, -2 (horizontāla taisne)
Sagaidāmais rezultāts: metode atgriež skaitli 1

4.1.5. Testpiemērs 5 – metode distanceFromLineToPoint(double x1, double y1, double Ax, double Ay, double Bx, double By)

Testpiemēra ID: s1t5
Apraksts: tiek izsaukta metode `distanceFromLineToPoint`, kurai padod kā parametrus 0, 1, 0, 2, 0, -2 (vertikāla taisne, punkts atrodas uz taisnes)
Sagaidāmais rezultāts: metode atgriež skaitli 0

4.1.6. Testpiemērs 6 – metode distanceFromLineToPoint(double x1, double y1, double Ax, double Ay, double Bx, double By)

Testpiemēra ID: s1t6
Apraksts: tiek izsaukta metode `distanceFromLineToPoint`, kurai padod kā parametrus -2, 3, -2, 3, -1, -2 (punkts sakrīt ar vienu no galapunktiem)
Sagaidāmais rezultāts: metode atgriež skaitli 0

Moduļa *SphericalWave* testpiemēru specifikācija

Dokumenta identifikators: Image.UT.TS.SPW.1.0
Projekta identifikators: Image
Moduļa identifikators: SPW
Autors: Jevgenijs Jonass

1. Klase UGraph

Visi nepieciešamie faili atrodas repozitorijā, mapē AtteluApstrade/testdata/graph. Katrā testpiemērā grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību. Kopu salīdzināšanai tiek lietota klases *HashSet* metode `equals`, kas strādā pateicoties pārdefinētām metodēm `hashCode` un `equals` klasēs *GraphEdge* un *Vertex*.

1.1. Testpiemēru kopa 1

1.1.1. Testpiemērs 1 – metode clone

Testpiemēra ID: s1t1
Nepieciešamie faili: graph1.in
Apraksts: grafs tiek klonēts ar metodi `clone`, pēc tam salīdzina grafu ar tā klonu.
Sagaidāmais rezultāts: metodes `getEdges` un `getVertices`, kas pielietotas abiem grafiem, atgriež vienādas kopas.

1.1.2. Testpiemērs 2 – metode get

Testpiemēra ID: s1t2
Nepieciešamie faili: graph1.in
Apraksts: tiek izsaukta metode `get(1, 1)`
Sagaidāmais rezultāts: metode atgriež virsotni ar koordinātām (1, 1) (virsotnes metodes `getX` un `getY` atgriež skaitli 1)

1.1.3. Testpiemērs 3 – metode addEdge

Testpiemēra ID: s1t3
Nepieciešamie faili: graph2.in
Apraksts: tiek izsaukta metode `addEdge (1, 2, 1, 1)`, kur koordinātām (1, 2) neatbilst neviena virsotne
Sagaidāmais rezultāts: metode atgriež false; virsotnes ar koordinātām (1, 1) pakāpe vienāda ar 0.

1.1.4. Testpiemērs 4 – metode neighbours

Testpiemēra ID: s1t4
Nepieciešamie faili: graph2.in
Apraksts: izsauc metodi `neighbours(3, 3)`
Sagaidāmais rezultāts: metode atgriež kopu {(2, 2), (5, 5), (6, 6), (7, 7)}

1.1.5. Testpiemērs 5 – metode deleteVertex

Testpiemēra ID: s1t5
Nepieciešamie faili: graph5.in
Apraksts: izsauc metodi `deleteVertex(4, 4)`
Sagaidāmais rezultāts: metode `getVertices` atgriež kopu {(1, 1), (2, 2), (3, 3)}; metode `getEdges` atgriež tukšo kopu

1.1.6. Testpiemērs 6 – metode edges

Testpiemēra ID: s1t6
Nepieciešamie faili: graph2.in
Apraksts: izsauc metodi `edges()`
Sagaidāmais rezultāts: metode atgriež skaitli 11

1.1.7. Testpiemērs 7 – metode clear

Testpiemēra ID: s1t7
Nepieciešamie faili: graph3.in
Apraksts: lokālajam mainīgajam S piešķir visu grafa virsotņu kopu; izsauc metodi `clear()`
Sagaidāmais rezultāts: metodes `getEdges` un `getVertices` atgriež tukšo kopu; visām virsotnēm kopā S laukiem `owner` un `neighbours` jābūt vienādam ar `null`.

1.1.8. Testpiemērs 8 – metode addVertex

Testpiemēra ID: s1t8
Nepieciešamie faili: graph4.in
Apraksts: izsauc metodi `addVertex(1, 1)`
Sagaidāmais rezultāts: metode atgriež `false`; metode `getVertices` atgriež kopu {(1, 1), (2, 2), (3, 3)}; metode `getEdges` atgriež tukšo kopu.

1.1.9. Testpiemērs 9 – metode getEdges

Testpiemēra ID: s1t9
Nepieciešamie faili: graph5.in
Apraksts: izsauc metodi `getEdges()`
Sagaidāmais rezultāts: metode atgriež kopu {((1, 1), (4, 4)), ((2, 2), (4, 4)), ((3, 3), (4, 4))}.

1.1.10. Testpiemērs 10 – metode getVertices

Testpiemēra ID: s1t10
Nepieciešamie faili: graph5.in
Apraksts: izsauc metodi `getVertices()`
Sagaidāmais rezultāts: metode atgriež kopu {(1, 1), (4, 4), (2, 2), (3, 3)}.

2. Klase Vertex

2.1. Testpiemēru kopa 1

2.1.1. Testpiemērs 1 – metode equals

Testpiemēra ID: s1t1
Apraksts: tiek izveidotas 2 jaunas virsotnes ar koordinātām (9, 16) un (20, 10), kas nepieder nevienam grafam. Pirmajai virsotnei tiek izsaukta metode equals, kurai padod kā parametru otro virsotni.
Sagaidāmais rezultāts: metode atgriež `false`.

2.1.2. Testpiemērs 2 – metode equals

Testpiemēra ID: s1t2
Apraksts: tiek izveidota jaunas virsotnes ar koordinātām (9, 16), kas nepieder nevienam grafam, un jauns `Object` tipa objekts. Pirmajai virsotnei tiek izsaukta metode equals, kurai padod kā parametru `Object` tipa objektu.
Sagaidāmais rezultāts: metode atgriež `false`.

2.1.3. Testpiemērs 3 – metode setLocation

Testpiemēra ID: s1t3
Nepieciešamie faili: graph1.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (2, 2) tiek pielietota metode `setLocation(9, 9)`
Sagaidāmais rezultāts: metode atgriež `true`; grafa metode `getVertices()` atgriež kopu {(1, 1), (3, 3), (4, 4), (9, 9)}; grafa metode getEdges() atgriež kopu {((1, 1), (9, 9)), ((4, 4), (9, 9)), ((3, 3), (9, 9)), ((3, 3), (4, 4))}

2.1.4. Testpiemērs 4 – metode degree

Testpiemēra ID: s1t4
Nepieciešamie faili: graph1.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (1, 1) tiek pielietota metode `degree()`.
Sagaidāmais rezultāts: metode atgriež skaitli 1

2.1.5. Testpiemērs 5 – metode adjacent

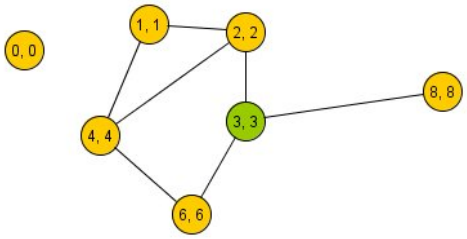
Testpiemēra ID: s1t5
Nepieciešamie faili: graph2.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (3, 3) tiek pielietota metode `adjacent`, kurai padod kā parametru virsotni ar koordinātām (5, 5)
Sagaidāmais rezultāts: metode atgriež `true`

2.1.6. Testpiemērs 6 – metode delete

Testpiemēra ID: s1t6
Nepieciešamie faili: graph3.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (4, 4) tiek pielietota metode `delete()`
Sagaidāmais rezultāts: metode atgriež `true`; grafa metode `getEdges` atgriež kopu {((1, 1), (3, 3)), ((3, 3), (2, 2))}; grafa metode `getVertices` atgriež kopu {(0, 0), (1, 1), (2, 2), (3, 3)}

2.1.7. Testpiemērs 7 – metode merge

Testpiemēra ID: s1t7
Nepieciešamie faili: graph2.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (3, 3) tiek pielietota metode `merge`, kurai padod kā parametrus virsotnes ar koordinātām (5, 5) un (7, 7)
Sagaidāmais rezultāts: metode atgriež `true`; grafa metode `getEdges` atgriež kopu {((1, 1), (2, 2)), ((1, 1), (4, 4)), ((2, 2), (4, 4)), ((2, 2), (3, 3)), ((4, 4), (6, 6)), ((3, 3), (6, 6)), ((3, 3), (8, 8))}; grafa metode `getVertices` atgriež kopu {(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (6, 6), (8, 8)}:



2.1.8. Testpiemērs 8 – metode merge

Testpiemēra ID: s1t8
Nepieciešamie faili: graph3.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; pēc tam virsotnei ar koordinātām (0, 0) tiek pielietota metode `merge`, kurai padod kā parametrus virsotnes ar koordinātām (4, 4), (1, 1) un (3, 3)
Sagaidāmais rezultāts: metode atgriež `true`; grafa metode `getEdges` atgriež kopu {((0, 0), (2, 2))}; grafa metode `getVertices` atgriež kopu {(0, 0), (2, 2)}

2.1.9. Testpiemērs 9 – metode merge

Testpiemēra ID: s1t9
Nepieciešamie faili: graph3.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; lokālajos mainīgajos E un V tiek saglabātas šķautņu un virsotņu kopas; pēc tam virsotnei ar koordinātām (0, 0) tiek pielietota metode `merge`, kurai padod kā parametrus tā paša grafa virsotnes ar koordinātām (4, 4), (1, 1) un jaunu virsotni ar koordinātām (3, 3), kura nepieder šim grafam
Sagaidāmais rezultāts: metode atgriež `false`; grafa metode `getEdges` atgriež kopu E; grafa metode `getVertices` atgriež kopu V

2.1.10. Testpiemērs 10 – metode merge

Testpiemēra ID: s1t10
Nepieciešamie faili: graph1.in
Apraksts: grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību; virsotnei ar koordinātām (4, 4) tiek pielietota metode `merge`, kurai padod kā parametrus virsotni ar koordinātām (1, 1) un šo pašu virsotni (ar koordinātām (4, 4))
Sagaidāmais rezultāts: metode atgriež `true`; grafa metode `getEdges` atgriež kopu {((1, 1), (2, 2)), ((2, 2), (3, 3)), ((3, 3), (4, 4)), ((2, 2), (4, 4)), ((1, 1), (4, 4))}; grafa metode `getVertices` atgriež kopu {(1, 1), (2, 2), (3, 3), (4, 4)}

3. Klase VertexIterator

3.1. Testpiemēru kopa 1

Tiek testēts, kā visas metodes darbojas kopīgi. Tiek pēc kārtas izsauktas metodes `next()` un `hasNext()`, kā arī konstruktors, saglabājot atgrieztās virsotnes kopā (*HashSet <Vertex>* objektā).

3.1.1. Testpiemērs 1

Testpiemēra ID: s1t1
Nepieciešamie faili: graph1.in
Sagaidāmais rezultāts: iegūtā kopa vienāda ar {(1, 1), (2, 2), (3, 3), (4, 4)}

3.1.2. Testpiemērs 2

Testpiemēra ID: s1t2
Nepieciešamie faili: graph6.in
Sagaidāmais rezultāts: iegūtā kopa ir tukša

4. Klase EdgeIterator

4.1. Testpiemēru kopa 1

Tiek testēts, kā visas metodes darbojas kopīgi. Tiek pēc kārtas izsauktas metodes `next()` un `hasNext()`, kā arī konstruktors, saglabājot atgrieztās virsotnes kopā (*HashSet <GraphEdge>* objektā).

4.1.1. Testpiemērs 1

Testpiemēra ID: s1t1
Nepieciešamie faili: graph1.in
Sagaidāmais rezultāts: iegūtā kopa vienāda ar {((1, 1), (2, 2)), ((2, 2), (3, 3)), ((2, 2), (4, 4)), ((3, 3), (4, 4))}

4.1.2. Testpiemērs 2

Testpiemēra ID: s1t2
Nepieciešamie faili: graph4.in
Sagaidāmais rezultāts: iegūtā kopa ir tukša

4.1.3. Testpiemērs 3

Testpiemēra ID: s1t3
Nepieciešamie faili: graph6.in
Sagaidāmais rezultāts: iegūtā kopa ir tukša

5. Klase GraphReader

5.1. Testpiemēru kopa 1 – metode read

Katrā testpiemērā tiek izveidots korekts *File* tipa objekts ar atbilstošo faila vārdu, ko padod metodei `read`.

5.1.1. Testpiemērs 1

Testpiemēra ID: s1t1
Nepieciešamie faili: graph1.in
Apraksts: failam ir korekts formāts
Sagaidāmais rezultāts: atgrieztais grafs nav `null`, tā metode `getEdges()` atgriež kopu {((1, 1), (2, 2)), ((2, 2), (3, 3)), ((2, 2), (4, 4)), ((3, 3), (4, 4))}, un metode `getVertices()` atgriež kopu {(1, 1), (2, 2), (3, 3), (4, 4)}

5.1.2. Testpiemērs 2 – nekorekts faila formāts

Testpiemēra ID: s1t2
Nepieciešamie faili: err1.in
Apraksts: failā ir uzdots negatīvs virsotņu skaits.
Sagaidāmais rezultāts: atgrieztais grafs ir `null`

5.1.3. Testpiemērs 3 – nekorekts faila formāts

Testpiemēra ID: s1t3
Nepieciešamie faili: err2.in
Apraksts: faila 4. rindā ir tikai 1 skaitlis, bet jābūt 2 skaitļiem.
Sagaidāmais rezultāts: atgrieztais grafs ir `null`

5.1.4. Testpiemērs 4 – nekorekts faila formāts

Testpiemēra ID: s1t4
Nepieciešamie faili: err3.in
Apraksts: failā ir uzdota nekorekta otra šķautne, jo pirmā tās virsotne neeksistē
Sagaidāmais rezultāts: atgrieztais grafs ir `null`

5.1.5. Testpiemērs 5 – nekorekts faila formāts

Testpiemēra ID: s1t5
Nepieciešamie faili: err4.in
Apraksts: failā ir uzdotas dublējošas šķautnes: otrā dublējas ar pirmo
Sagaidāmais rezultāts: atgrieztais grafs ir `null`

5.2. Testpiemēru kopa 2 – metode write

Visos testpiemēros grafs (G1) tiek nolasīts no faila (F1) ar metodi `read`, klonēts ar metodi `clone()` un ierakstīts citā failā (F2) ar metodi `write`. Pēc tam grafs tiek nolasīts no faila (F2) ar metodi `read()`, salīdzināts ar grafu (G2) ar metožu `getEdges()` un `getVertices()` palīdzību, salīdzinot kopas, ko atgriež šīs metodes. Kopām, ko atgriež grafa G1 metodes, jābūt vienādām ar kopām, ko atgriež grafa G2 metodes.

5.2.1. Testpiemērs 1

Testpiemēra ID: s2t1
F1: graph1.in
F2: graph1.tmp

5.2.2. Testpiemērs 2

Testpiemēra ID: s2t2
F1: graph2.in
F2: graph2.tmp

5.2.3. Testpiemērs 3

Testpiemēra ID: s2t3
F1: graph3.in
F2: graph3.tmp

5.2.4. Testpiemērs 4

Testpiemēra ID: s2t4
F1: graph4.in
F2: graph4.tmp

5.2.5. Testpiemērs 5

Testpiemēra ID: s2t5
F1: graph5.in
F2: graph5.tmp

5.2.6. Testpiemērs 6

Testpiemēra ID: s2t6
F1: graph6.in
F2: graph6.tmp

6. Klase Optimization

Visos testpiemēros grafs tiek nolasīts no faila ar klases *GraphReader* palīdzību. Tiek iegūtas virsotņu kopa V un šķautņu kopa E ar metožu `getVertices` un `getEdges` palīdzību.

6.1. Testpiemēru kopa 1

6.1.1. Testpiemērs 1 – metode connectEnds

Testpiemēra ID: s1t1
Nepieciešamie faili: graph3.in
Apraksts: tiek izsaukta metode `connectEnds`, kurai padod kā parametrus grafu un attālumu 10
Sagaidāmais rezultāts: E={((0, 0), (4, 4)), ((4, 4), (1, 1)), ((1, 1), (3, 3)), ((3, 3), (2, 2)), ((0, 0), (2, 2))}, V={(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)}

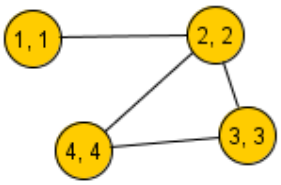
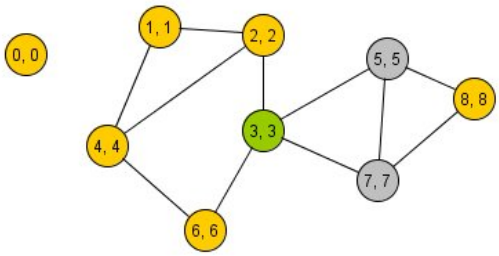
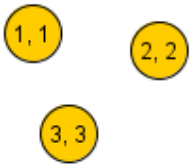
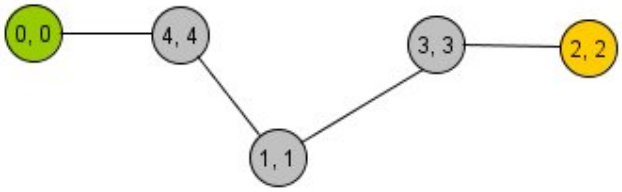
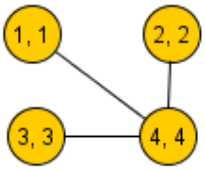
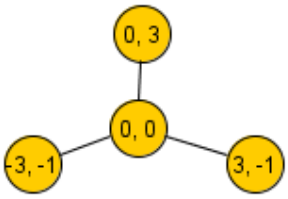
6.1.2. Testpiemērs 2 – metode cutTails

Testpiemēra ID: s1t2
Nepieciešamie faili: graph5.in
Apraksts: tiek izsaukta metode `cutTails`, kurai padod kā parametrus grafu un attālumu 2.9
Sagaidāmais rezultāts: E={((1, 1), (4, 4))}, V={(1, 1), (4, 4)}

6.1.3. Testpiemērs 3 – metode primaryOptimization

Testpiemēra ID: s1t3
Nepieciešamie faili: graph7.in
Apraksts: tiek izsaukta metode `primaryOptimization`, kurai padod kā parametrus grafu un slieksni 1
Sagaidāmais rezultāts: E={((0, 3), (0, -1)), ((0, -1), (-3, -1)), ((0, -1), (3, -1))}, V={(0, -1), (3, -1), (-3, -1), (0, 3)}

7. Pielikums – grafu specifikācija

<table>
<tr><td>graph1.in
 
</td><td>graph2.in
 
</td></tr>
<tr><td>graph4.in
 
</td><td>graph3.in
 
</td></tr>
<tr><td>graph5.in
 
</td><td>graph7.in
 
</td></tr>
<tr><td>graph6.in: tukšais grafs</td><td></td></tr>
</table>

Testēšanas žurnāls

1. Apraksts

Šajā testēšanas žurnālā ir aprakstīta programmas "AttēluApstrāde" testēšana, kas veikta testēšanas plāna Image.UT.TP.1.0 ietvaros.

2. Saistītie dokumenti

- PPS: Image.PPS.1.0
- PPA: Image.PPA.1.0
- Projekta "AttēluApstrāde" *Javadoc* dokumentācija
- Testēšanas plāns: Image.UT.TP.1.0

3. Darbību un notikumu pieraksts

Testējamā klase	Testu ID	Veiksmīgi	TPZ	Datums
math.CoordVector	s1t1 – s1t5	☒ Jā ☐ Nē		15.04.2009
math.Line	s1t1 – s1t6	☒ Jā ☐ Nē		15.04.2009
graph.Vertex	s1t1 – s1t10	☒ Jā ☐ Nē		22.04.2009
alg.SortedArray	s1t1	☐ Jā ☒ Nē	TPZ.01	02.05.2009
graph.EdgeIterator	s1t1 – s1t3	☒ Jā ☐ Nē		04.05.2009
graph.GraphReader	s1t1 – s1t5	☒ Jā ☐ Nē		04.05.2009
graph.UGraph	s1t1 – s1t10	☒ Jā ☐ Nē		04.05.2009
graph.VertexIterator	s1t1 – s1t2	☒ Jā ☐ Nē		04.05.2009
alg.SortedArray	s1t1 – s1t8	☒ Jā ☐ Nē		13.05.2009
alg.MedianFilter	s1t1 – s1t3	☒ Jā ☐ Nē		13.05.2009
alg.SortedArray	s2t1 – s2t10	☒ Jā ☐ Nē		15.05.2009
graph.GraphReader	s2t1 – s2t6	☒ Jā ☐ Nē		15.05.2009
spw.Optimization	s1t1 – s1t3	☒ Jā ☐ Nē		15.05.2009

4. Testu problēmu ziņojumi

4.1. Ziņojums TPZ.01

Testējamā klase: *SortedArray*
Testpiemēra ID: s1t1
Statuss: izlabots
Problēmas apraksts: metode `average()` izdala elementu summu ar elementu skaitu, bet nepārveido tipu par `double`. Tā rezultātā faktiski notiek rezultāta noapaļošana uz leju.

Projekta organizācija

Projekts "AttēluApstrāde" tika veidots pilnīgi no nekā. Tajā piedalījās 9 LU studenti prakses ietvaros. Projekts tika sadalīts moduļos, kurus izstrādāja katrs atsevišķi. Projekta sākumā tika apgūti Java valodas pamatprincipi un attēlu apstrādes teorijas pamati. Pēc tam tika izstrādāta prasību specifikācija un iesākta programmatūras projektējuma izveide. Izstrādes laikā tika veikta vienībtestēšana.

Kvalitātes nodrošināšana

Lai nodrošinātu kvalitāti, izstrādājamajai programmatūrai tika uzrakstīta dokumentācija – Programmatūras prasību specifikācija un Programmatūras projektējuma apraksts atbilstoši Latvijas Nacionālais standartizācijas un metroloģijas centra izstrādātajiem Latvijas valsts standartiem. Kods ir rakstīts saskaņā ar labo programmēšanas praksi un atbilstoši komentēts.

Konfigurāciju pārvaldība

Konfigurāciju pārvaldībai tika izmantots konfigurāciju vadības rīks Subversion, kas ļauj vairākiem programmētājiem rediģēt dažādas projekta daļas, netraucējot citus. Kad kādas daļas pievienošana vai rediģēšana ir paveikta, to var augšupielādēt uz servera, kurš saglabā katras izmaiņas. Tiek glabāta arī izmaiņu vēsture, tā ka, ja, veicot izmaiņas, sanāk sabojāt kodu, ir iespēja atgriezties pie strādājošas versijas. Tas izslēdz iespēju, veicot mazus labojumus kādā sīkā projekta modelī, sabojāt visu projektu.

Darbietilpības novērtējums

Darbietilpības novērtēšanai izvēlējos COCOMO modeli.

SF1	4	Nav pieredzes šādu uzdevumu veikšanā.
SF2	1	Darbu nedaudz ierobežo laiks.
SF3	2	Daži izstrādāti algoritmi netika aprakstīti nekur.
SF4	2	Savstarpēja komunikācija bija stipri nepieciešama un nedaudz apgrūtināta
SF5	4	Projekta izstrādes procesa ļoti vāja organizācija.

B = 1.01 + 0.01*13 = 1.01 + 0.13 = **1.14**

EM1	1.4	Ļoti zemas izstrādātāja spējas
EM 2.1 – 2.3	1.0	Neliels uzsvars uz dokumentēšanu.
	1.2	Vidēja produkta sarežģītība
	0.8	Datu bāzes nav.
EM3	1.2	Potenciāli liela daļa no izstrādātu funkciju var būt izmantota citos moduļus, bet reāli projektā tiek izmantotas citas realizācijas.
EM4	1.1	Programmas izpildes laiks nav ierobežots ar definētu sekunžu

		skaitu.
EM 5.1-5.3	1	Izstrādes vide diezgan stabila, bet izstrādes laikā gadījās problēmas.
	0.8	Labi integrēta vide produkta izstrādei.
	0.8	Ļoti labs atbalsts programmatūras izstrādes komandā.
EM6	1.1	Izstrādes kalendārais laiks nav pārāk saspiests.

E = 1.25

SLOC = 1037

PM = 2.5*1.25*(0.001*1037)^1.14= **3.2**

Darba izstrāde noritēja 3 mēnešu laikā, prakses ietvaros. Daļa no prakses laika tika veltīta Java valodas apgūšanai, izstrādes vides iepazīšanai un pamatalgoritmu izpētei attēlu apstrādes sfērā.

Rezultāti

Attēlu apstrādes programma un visi tās moduļi tika veiksmīgi izstrādāti un implementēti, kā arī tika izveidota nepieciešamā dokumentācija. Projekta galvenā daļa bija sistēmas arhitektūras projektēšana no paša sākuma. Paralēli notika algoritmu projektēšana, kā arī lietotāja saskarnes veidošana. Lai pārliecinātos par programmatūras pareizu darbību, projekta beigās tika veikta vienībtestēšana, kas tika arī atbilstoši dokumentēta.

Secinājumi

Programmu "AttēluApstrāde", kā arī moduli *SphericalWave*, turpmāk varēs attīstīt tālāk, palielinot funkciju skaitu un sarežģītību. Izstrādājot darbu, apguvu programmēšanas valodu Java, labas kodēšanas prakses pamatprincipus, un programmatūras dokumentācijas izstrādi, kā arī attēlu apstrādes pamatus, uzlaboju zināšanas par grafu teoriju un grafu algoritmiem.

Izmantotā literatūra

http://ocrai.narod.ru/vectory.html – Применение волнового алгоритма для нахождения скелета растрового изображения

http://www.shellandslate.com/download/fastmedian_5506.pdf - Fast Median and Bilateral Filtering

http://djvu-soft.narod.ru/bookscanlib/project.htm – Реализация проекта BookScanLib

Pielikums 1

Sekojošā tabula parāda, kuras galvenā moduļa daļas autors ir izstrādājis (+ nozīmē pilnībā visu, – nozīmē neko):

Klases

Pakotne *alg*	
Spline	+
Binarization	+
Bresenham	+
MedianFilter	+
MeanFilter	+
SortedArray	+
Pakotne *gui*	
AbstractFrame	+
MainFrame	daļēji
MFAListener	daļēji
FilterFrame	+
BinarizationFrame	+
ResizingFrame	+
ImageViewer	+
EditToolBar	+
SwingImageLoader	+
... (citu moduļu logi)	–
Pakotne *util*	
ImageProcessor	daļēji
RasterImage	daļēji
Pair	+
Triplet	+
UnorderedPair	+
PointSet	+
Command	+
PostTaglet	+
PreTaglet	+
Settings	+
Pakotne *math*	
CoordVector	+
Line	+

Funkcijas

Identifikators	
Func.Edit.1.Gray	–
Func.Edit.2.RotateACW	+
Func.Edit.3.RotateCW	+
Func.Edit.4.HorFlip	–
Func.Edit.5.VertFlip	+
Func.Edit.6.Negative	–
Func.Edit.7.Crop	+
Func.Edit.8.Undo	+
Func.Edit.9.Redo	+
Func.Edit.10.frw	+
Func.Edit.11.ffw	+
Func.File.1.ImgOpen	–
Func.File.2.ImgSave	+
Func.File.3.ImgClose	+
Func.Alg.1.Bernsen	+
Func.Alg.2.Otsu	+
Func.Alg.3.binthr	+
Func.Alg.4.rsmnn	–
Func.Alg.5.rsmbil	+
Func.Alg.6.varfilter	+
Func.Alg.7.medfilbid	+
Func.Alg.8.medfilmtrx	+
Func.Alg.9.Niblack	+
Func.Alg.10.mnfltrsqr	+

Pielikums 2 – Javadoc

Visam projektam tika uzģenerēta *Javadoc* dokumentācija, kas apraksta visas pakotnes, klases un interfeisus, klašu un interfeišu metodes un klašu laukus. Tā kā visa dokumentācija aizņem daudz vietas, šeit tika ievietota dokumentācija klasēm *RasterImage* un *SortedArray*. Dokumentācijā ir aprakstīti arī klašu invarianti, jo šajā projektā dokumentācija nav domāta tikai saskarnes aprakstam.

alg

Class SortedArray

`java.lang.Object`

`alg.SortedArray`

`public final class` **`SortedArray`**
`extends java.lang.Object`

Class representing sorted array of constant size. Keeps track of the order in which elements were added and allows to replace the oldest element in the array with specified element.

Class invariant:

1. arr != null
2. q != null
3. arr.length > 0
4. arr.length == q.size()
5. arr is sorted in ascending order (equal elements are allowed)
6. for each element e in q: [e != null and e.intValue() is an element of `arr`]
7. for each element e in arr: [q.contains(e)]
8. elements in q are queued in the same order as they were added to this array: top element in q was added before all other elements
9. `sum` is equal to the sum of elements in `arr`

Author:
Jevgenijs Jonass.

Since:
01.05.2009

Constructor Summary

`SortedArray``(int len)` Constructs an array of `len` elements.

Method Summary

`double`	**`average`**`()` Returns average of the elements.
`int`	**`getElement`**`(int pos)` Returns element at specified position in this sorted array.
`int`	**`getMedian`**`()` Returns the median - element at index length/2.

void	**init**(int[] buffer, int start, int stepSize, int width, int height, int shift) Initializes the array with [width*height] elements from `buffer`, which represents a 2D array of pixels.
void	**replace**(int m) Replaces the oldest element in the array with specified element.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SortedArray

`public` **SortedArray**(int len)

Constructs an array of `len` elements.

Parameters:

`len` - length of the array. Cannot be changed after the array is created.

Author:

Jevgenijs Jonass.

Since:

01.05.2009

Preconditions:

1. `len%gt;0.`

Postconditions:

1. `class invariant of this is true`
2. `this array is filled with 0`

Method Detail

getElement

`public int` **getElement**(int pos)

Returns element at specified position in this sorted array.

Parameters:

`pos` - position of the element.

Returns:

array element.

Author:

Jevgenijs Jonass.

Since:

01.05.2009

Preconditions:

1. `pos >= 0 and pos < length of this array`
2. `class invariant of this is true`

Postconditions:

1. `this remains unchanged`

---

getMedian

```
public int getMedian()
```

Returns the median - element at index length/2.

Returns:
the median.

Author:
Jevgenijs Jonass.

Since:
01.05.2009

Preconditions:

1. `class invariant of this is true`

Postconditions:

1. `this remains unchanged`

---

average

```
public double average()
```

Returns average of the elements.

Returns:
average of the elements.

Author:
Jevgenijs Jonass.

Since:
05.05.2009

Preconditions:

1. `class invariant of this is true`

Postconditions:

1. `this remains unchanged`

---

init

```
public void init(int[] buffer,
                 int start,
                 int stepSize,
                 int width,
                 int height,
                 int shift)
```

Initializes the array with [width*height] elements from `buffer`, which represents a 2D array of pixels. Elements are taken from rectangular area, whose left top pixel has index

`start`. Element values are shifted by `shift` bits to the right and then mask 0x000000ff is applied.

Order in which elements are added: from top to bottom, from left to right

Parameters:
`buffer` - source buffer (ownership: caller)
`start` - index of the first pixel in the rectangular area
`stepSize` - distance between one row of pixels and the next row in `buffer` (width of the image)
`width` - width of the rectangular area of pixels
`height` - height of the rectangular area of pixels
`shift` - Bit shift (0, 8 or 16)
Author:
Jevgenijs Jonass.
Since:
01.05.2009
Preconditions:

1. `class invariant of this is true`
2. `buffer != null`
3. `start %gt;= 0 and start+height*stepSize <= buffer.length`
4. `stepsize %gt;= width %gt;= 0`
5. `width*height == [length of this array]`

Postconditions:

1. `class invariant of this is true`
2. `buffer will not be modified.`

---

replace

`public void` **`replace`**`(int m)`

Replaces the oldest element in the array with specified element.
Parameters:
`m` - the new element.
Author:
Jevgenijs Jonass.
Since:
01.05.2009
Preconditions:

1. `class invariant of this is true`

Postconditions:

1. `class invariant of this is true`

util

Class RasterImage

```
java.lang.Object
```

util.RasterImage

All Implemented Interfaces:

java.lang.Cloneable

```
public final class RasterImage
extends java.lang.Object
implements java.lang.Cloneable
```

Class representing raster image.

Author:

Karlis Freivalds.

Since:

03.02.2009

Field Summary

int	**height** Height of the image.
int[]	**pixels** Represents a two-dimensional array of pixels.
int	**width** Width of the image.

Constructor Summary

RasterImage(int width, int height) Constructor that creates image with specified width and height.
RasterImage(RasterImage rImage) Constructor that creates a copy of the image.

Method Summary

java.lang.Object	**clone**() Clones the image.
void	**copyTo**(RasterImage target) Copies this image to the destination image.
boolean	**equals**(java.lang.Object obj) Determines whether or not two images are equal.
java.awt.image.ImageProducer	**getSource**() Returns image producer.
int	**greyValueAt**(int i) Returns intensity of the pixel at specified index.
int	**greyValueAt**(int x, int y) Returns intensity of the pixel at specified coordinates

	using default kr, kg and kb.
int	**hashCode**()
boolean	**in_bounds**(java.awt.Rectangle selection) Checks if selection is in bounds of this image.
boolean	**is_binary**() Checks if the image is binary (pixel values are either 0 or 0x00ffffff).
boolean	**is_grayscale**() Checks if the image is grayscale (RGB component values are equal).
void	**paintImage**(java.awt.Graphics paintOn, int offsetX, int offsetY) **Deprecated.**
void	**setPixel**(int x, int y, int value) Sets pixel value.
java.awt.image.BufferedImage	**toBuffered**() Converts RasterImage to BufferedImage of type BufferedImage.TYPE_INT_RGB.
java.awt.image.BufferedImage	**toGrayscaleBuffered**() Converts RasterImage to BufferedImage of type BufferedImage.TYPE_BYTE_GRAY.
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

pixels

public int[] **pixels**

Represents a two-dimensional array of pixels. First dimension corresponds to the horizontal axis x (from left to the right) and can have values 0..width-1. Second dimension corresponds to the vertical axis y (from top to bottom) and can have values 0..height-1.

Pixel array must have width*height elements. Pixel at coordinate [i, j] can be accessed as pixels[width*j+i].

Pixel values should lie in range [0..0x00ffffff]. Pixel values have format 0x00rrggbb (alpha is not present, the highest byte must always be 0).

Ownership of pixels: client. For all RasterImage objects r1, r2, such that r1!=r2, r1.pixels != r2.pixels. Array pixels can be recreated by any method.

Author:
Karlis Freivalds.

Since:
03.02.2009

---

width

`public int` **`width`**

Width of the image. Must be positive integer.

Author:
Karlis Freivalds.

Since:
03.02.2009

---

height

`public int` **`height`**

Height of the image. Must be positive integer.

Author:
Karlis Freivalds.

Since:
03.02.2009

Constructor Detail

RasterImage

```
public RasterImage(int width,
                   int height)
```

Constructor that creates image with specified width and height. Initializes pixels with 0.

Parameters:

`width` - Width of the image.

`height` - Height of the image.

Author:
Jevgenijs Jonass.

Since:
18.03.2009

Preconditions:

1. `width and height are positive (cannot be 0)`

Postconditions:

1. `this.invariant()==true`
2. `this.width==width, this.height==height`

RasterImage

`public` **`RasterImage`**`(RasterImage rImage)`

Constructor that creates a copy of the image.

Parameters:

`rImage` - The image that must be copied (ownership: caller)

Author:
Jevgenijs Jonass.

Since:
18.03.2009
Preconditions:

1. `rImage!=null`
2. `rImage.invariant()==true`

Postconditions:

1. `this.invariant()==true`
2. `rImage remains unchanged.`
3. `this.equals(rImage)`

Method Detail

clone

`public java.lang.Object` **`clone`**`()`

Clones the image. Allocates memory for the resulting image.
Overrides:
`clone` in class `java.lang.Object`
Returns:
Cloned RasterImage object (ownership: caller)
Author:
Jevgenijs Jonass.
Since:
17.04.2009
Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`
2. `(returned image).invariant()==true`
3. `this.equals(returned image)`

---

equals

`public boolean` **`equals`**`(java.lang.Object obj)`

Determines whether or not two images are equal. Two instances of `RasterImage` are equal if they have equal width and height and the values of their pixels are the same.
Overrides:
`equals` in class `java.lang.Object`
Parameters:
`obj` - an object to be compared with this `RasterImage` (ownership: caller)
Returns:
`true` if the object to be compared is an instance of `RasterImage` and has the same size and pixel values.
Author:
Jevgenijs Jonass.

Since:
12.05.2009
Preconditions:

1. `obj is instance of RasterImage => class invariant of obj is true`
2. `class invariant of this is true`

Postconditions:

1. `this remains unchanged`
2. `obj remains unchanged`

---

hashCode

`public int` **`hashCode`**`()`

Overrides:
`hashCode` in class `java.lang.Object`
Returns:
A hash code for this `RasterImage`.
Author:
Jevgenijs Jonass.
Since:
12.05.2009
Preconditions:

1. `class invariant of this is true`

Postconditions:

1. `this remains unchanged`

---

toString

`public java.lang.String` **`toString`**`()`

Overrides:
`toString` in class `java.lang.Object`
Returns:
a string representation of this image.
Author:
Jevgenijs Jonass.
Since:
12.05.2009
Preconditions:

1. `class invariant of this> is true`

Postconditions:

1. `this remains unchanged`

is_binary

`public boolean` **`is_binary`**`()`

Checks if the image is binary (pixel values are either 0 or 0x00ffffff).

Returns:
true if the image is binary.

Author:
Jevgenijs Jonass.

Since:
16.04.2009

Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

---

is_grayscale

`public boolean` **`is_grayscale`**`()`

Checks if the image is grayscale (RGB component values are equal).

Returns:
true if the image is grayscale.

Author:
Jevgenijs Jonass.

Since:
16.04.2009

Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

---

toBuffered

`public java.awt.image.BufferedImage` **`toBuffered`**`()`

Converts RasterImage to BufferedImage of type BufferedImage.TYPE_INT_RGB.

Returns:
`BufferedImage` of the same size (ownership: caller)

Author:
Ainars Kumpins

Since:
23.03.2009

Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

---

toGrayscaleBuffered

```
public java.awt.image.BufferedImage toGrayscaleBuffered()
```

Converts RasterImage to BufferedImage of type BufferedImage.TYPE_BYTE_GRAY. Gray value for each pixel is calculated using default kr, kg and kb.

Returns:

`BufferedImage` of the same size (ownership: caller)

Author:

Jevgenijs Jonass.

Since:

14.05.2009

Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

---

greyValueAt

```
public int greyValueAt(int x,
                       int y)
```

Returns intensity of the pixel at specified coordinates using default kr, kg and kb.

Parameters:

`x` - The horizontal coordinate.

`y` - The vertical coordinate.

Returns:

grey value (intensity).

Author:

Ainārs Kumpiņš.

Since:

08.03.2009

Preconditions:

1. `this.invariant()==true`
2. `x is in range [0..width-1] and y is in range [0..height-1].`

Postconditions:

1. `this image remains unchanged`

---

greyValueAt

```
public int greyValueAt(int i)
```

Returns intensity of the pixel at specified index.
Parameters:
`i` - The array index.
Returns:
intensity in range [0..255].
Author:
Ainārs Kumpiņš.
Since:
08.03.2009
Preconditions:

1. `this.invariant()==true`
2. `i is in range [0..width*height-1].`

Postconditions:

1. `this image remains unchanged`

---

setPixel

```
public void setPixel(int x,
                     int y,
                     int value)
```

Sets pixel value.
Parameters:
`x` - The horizontal coordinate.
`y` - The vertical coordinate.
`value` - Pixel value.
Author:
Jevgenijs Jonass.
Since:
04.04.2009
Preconditions:

1. `this.invariant()==true`
2. `highest byte of value is 0`
3. `x is in range [0..width-1] and y is in range [0..height-1]`
4. `highest byte of value is 0.`

Postconditions:

1. `this.invariant()==true`
2. `this.pixels[y*width+height]==value`

---

copyTo

```
public void copyTo(RasterImage target)
```

Copies this image to the destination image. Destination image may have different size than `this` image. `source and this` may point to the same objects.
Parameters:

`target` - Target image (ownership: caller)
Author:
Jevgenijs Jonass.
Since:
04.04.2009
Preconditions:

1. `target != null`
2. `this.invariant()==true`
3. `target.invariant()==true`

Postconditions:

1. `this.width == target.width and this.height == target.height`
2. `target.invariant()==true`
3. `this.equals(target)`
4. `this image remains unchanged`

---

paintImage

```
@Deprecated
public void paintImage(java.awt.Graphics paintOn,
                       int offsetX,
                       int offsetY)
```

Deprecated. *22.03.2009 replaced with getSource() and ImageViewer class*
Paints this image.
Parameters:
`paintOn` - The graphics canvas to paint on (ownership: caller)
`offsetX` - Horizontal offset (any integer value).
`offsetY` - Vertical offset (any integer value).
Author:
Karlis Freivalds.
Since:
03.02.2009
Version:
20.02.2009 Jevgenijs Jonass - adde offsetX and offsetY parameters.
Preconditions:

1. `paintOn != null.`
2. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

Deprecated:
22.03.2009 replaced with getSource() and ImageViewer class

---

in_bounds

```
public boolean in_bounds(java.awt.Rectangle selection)
```

Checks if `selection` is in bounds of this image.
Parameters:
`selection` - rectangular area. Cannot be null. (ownership: caller)
Returns:
true if selection.width >= 0, selection.height >= 0, width > selection.x >= 0, height > selection.y >= 0, width >= selection.x+selection.width, height >= selection.y+selection.height
Author:
Jevgenijs Jonass.
Since:
19.05.2009
Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`
2. `Does not modify selection or`

---

getSource

`public java.awt.image.ImageProducer` **`getSource`**`()`

Returns image producer.
Returns:
The image producer (ownership: this RasterImage)
Author:
Jevgenijs Jonass.
Since:
22.03.2009
Preconditions:

1. `this.invariant()==true`

Postconditions:

1. `this image remains unchanged`

Pielikums 3 – programmas kods

Sakarā ar to, ka programmas pilnais kods ir ļoti apjomīgs, šajā nodaļā tiek ievietots dažu klašu kods.

1. Klase UGraph

```
package graph;

import java.util.*;

/**
 * Class representing non-oriented graph.
 *
 * <p>Class invariant:<ol>
 * <li>vertices is not null.</li>
 * <li>edges is equal to the number of edges in the graph (edges&gt;=0).</li>
 * <li>For each v in vertices [v.owner == this]</li>
 * <li>For each v, w in vertices [v!=w =&gt; v.neighbours !=
w.neighbours]</li>
 * <li>Class invariant is true for each v in vertices</li>
 * </ol></p>
 *
 * @author Jevgenijs Jonass.
 * @since 25.02.2009
 */
public final class UGraph implements Cloneable {

	/**
	 * Number of edges in the graph.
	 * @author Jevgenijs Jonass.
	 * @since 25.02.2009
	 */
	int edges;

	/**
	 * Vertices of the graph.
	 * @author Jevgenijs Jonass.
	 * @since 25.02.2009
	 */
	Hashtable <Vertex, Vertex> vertices;

	/**
	 * Constructor that creates an empty graph.
	 * @post class invariant of <code>this</code> is true
	 * @author Jevgenijs Jonass.
	 * @since 25.02.2009
	 */
	public UGraph() {
		vertices=new Hashtable <Vertex, Vertex> ();
		edges=0;
	}

	/**
	 * Creates a deep copy of this graph. Allocates memory for
	 * <code>vertices</code> and for all objects in it. The returned graph
	 * contains vertices at the same coordinates as vertices in this graph
and
	 * has the same edges.
	 *
	 * @pre class invariant of <code>this</code> is true
	 * @post class invariant of <code>this</code> is true
	 * @post class invariant of the returned <code>UGraph</code> is true
```

```
	 * @author Jevgenijs Jonass.
	 * @return Cloned object. Cannot be null. (ownership: caller)
	 * @since 17.04.2009
	 */
	@Override
	public Object clone() {
		UGraph g=new UGraph();
		VertexIterator it=vertices();
		while (it.hasNext()) {
			Vertex v=it.next();
			g.addVertex(v.getX(), v.getY());
		}
		EdgeIterator edgeIterator=new EdgeIterator(this);
		while (edgeIterator.hasNext()) {
			GraphEdge edge=edgeIterator.next();
			g.addEdge(edge.first.getX(), edge.first.getY(),
					edge.second.getX(), edge.second.getY());
		}
		return g;
	}

	/**
	 * Clears the graph (deletes all vertices and edges).
	 * @pre class invariant of <code>this</code> is true
	 * @post class invariant of <code>this</code> is true
	 * @author Jevgenijs Jonass.
	 * @since 19.04.2009
	 */
	public void clear() {
		VertexIterator it=vertices();
		while (it.hasNext()) {
			Vertex v=it.next();
			v.owner=null;
			v.neighbours=null;
		}
		vertices.clear();
		edges=0;
	}

	/**
	 * Creates and adds a vertex at specified coordinates and returns
pointer
	 * to the newly allocated Vertex object. If the graph already contains
	 * vertex at given coordinates, no changes will be made and false will
be
	 * returned. If the vertex was successfully added, it will have no
	 * neighbours after the method returns.
	 *
	 * @param x The x coordinate of the vertex.
	 * @param y The y coordinate of the vertex.
	 * @pre class invariant of <code>this</code> is true
	 * @post class invariant of <code>this</code> is true
	 * @post returned Vertex is not null =&gt; class invariant of returned
	 *		Vertex is true
	 * @return pointer to a Vertex object or <tt>null</tt> if the vertex was
not
	 *		added. (ownership: caller)
	 * @author Jevgenijs Jonass.
	 * @since 25.02.2009
	 */
	public Vertex addVertex(int x, int y) {
		Vertex v=new Vertex(x, y);
		if (vertices.containsKey(v)) return null;
		v.neighbours=new LinkedList <Vertex> ();
```

```
            v.owner=this;
            vertices.put(v, v);
            return v;
      }

      /**
       * Deletes specified vertex from this graph. If this graph does not
contain
       * specified vertex, no changes will be made.
       * @param v the vertex. Cannot be null. (ownership: caller)
       * @pre class invariant of <code>this</code> is true
       * @pre class invariant of <code>v</code> is true
       * @post class invariant of <code>v</code> is true
       * @post class invariant of <code>this</code> is true
       * @return true if the graph changed as a result of the call.
       * @author Jevgenijs Jonass.
       * @since 25.02.2009
       */
      public boolean deleteVertex(Vertex v) {
            assert !(v==null);
            if (v.owner!=this) return false;
            return v.delete();
      }

      /**
       * Returns the total number of edges in the graph.
       * @pre class invariant of <code>this</code> is true
       * @post <code>this</code> remains unchanged
       * @return Total number of edges in the graph.
       * @author Jevgenijs Jonass.
       * @since 25.02.2009
       */
      public int edges() {
            return edges;
      }

      /**
       * Returns the number of vertices in the graph.
       * @pre class invariant of <code>this</code> is true
       * @post <code>this</code> remains unchanged
       * @return Total number of vertices in the graph.
       * @author Jevgenijs Jonass.
       * @since 25.02.2009
       */
      public int size() {
            return vertices.size();
      }

      /**
       * Finds vertex at specified coordinates.
       * @param x The x coordinate of the vertex.
       * @param y The y coordinate of the vertex.
       * @pre class invariant of <code>this</code> is true
       * @post <code>this</code> remains unchanged
       * @post returned Vertex is not null =&gt; class invariant of returned
       *          Vertex is true
       * @post returned Vertex is not null =&gt; [returned Vertex].owner==this
       * @return pointer to a vertex or null if the graph does not contain
vertex
       *          at given coordinates. (ownership: caller)
       * @author Jevgenijs Jonass.
       * @since 18.04.2009
       */
      public Vertex get(int x, int y) {
```

```
            return vertices.get(new Vertex(x, y));
        }

        /**
         * Returns the iterator through all vertices in this graph.
         * The iterator does not allow structural modification of the vertex
set,
         * but you can call <code>setLocation</code> method for any vertex.
         * If you add/delete vertices from the graph while iterating through
         * the vertices, behaviour of the iterator methods is undefined
         * (it is not guaranteed that they will raise an exception), but you can
         * add/delete edges.
         *
         * <p>The returned VertexIterator will return pointers to Vertex objects
         * that belong to this.vertices (and not to their copies).</p>
         *
         * @pre class invariant of <code>this</code> is true
         * @post <code>this</code> remains unchanged
         * @return Vertex iterator. Cannot be null. (ownership: caller)
         * @author Jevgenijs Jonass.
         * @since 15.03.2009
         */
        public VertexIterator vertices() {
            return new VertexIterator(vertices.keySet().iterator());
        }

        /**
         * Finds two vertices at specified coordinates and
         * adds an edge that connects them. If this graph does not contain any
         * of the vertices or if they are adjacent, no changes will be made.
         *
         * @param x1 X coordinate of the first vertex.
         * @param y1 Y coordinate of the first vertex.
         * @param x2 X coordinate of the second vertex.
         * @param y2 Y coordinate of the second vertex.
         * @pre class invariant of <code>this</code> is true
         * @post class invariant of <code>this</code> is true
         * @return true if the graph changed as a result of the call.
         * @author Jevgenijs Jonass.
         * @since 25.02.2009
         */
        public boolean addEdge(int x1, int y1, int x2, int y2) {
            Vertex v=get(x1, y1);
            if (v==null) return false;
            return v.addNeighbour(x2, y2);
        }

        /**
         * Finds two vertices at specified coordinates and checks if they are
         * adjacent.
         * @param x1 X coordinate of the first vertex.
         * @param y1 Y coordinate of the first vertex.
         * @param x2 X coordinate of the second vertex.
         * @param y2 Y coordinate of the second vertex.
         * @pre class invariant of <code>this</code> is true
         * @post <code>this</code> remains unchanged
         * @return true if this graph contains both vertices and they are
adjacent.
         * @author Jevgenijs Jonass.
         * @since 25.02.2009
         */
        public boolean adjacent(int x1, int y1, int x2, int y2) {
            Vertex v1=get(x1, y1);
            if (v1==null) return false;
```

```
            Vertex v2=get(x2, y2);
            if (v2==null) return false;
            return v1.adjacent(v2);
      }

      /**
       * Finds two vertices at specified coordinates and
       * deletes edge that connects them. If this graph does not contain any
       * of the vertices or if they are not adjacent, no changes will be made.
       *
       * @param x1 X coordinate of the first vertex.
       * @param y1 Y coordinate of the first vertex.
       * @param x2 X coordinate of the second vertex.
       * @param y2 Y coordinate of the second vertex.
       * @pre class invariant of <code>this</code> is true
       * @post class invariant of <code>this</code> is true
       * @return true if the graph changed as a result of the call.
       * @author Jevgenijs Jonass.
       * @since 25.02.2009
       */
      public boolean deleteEdge(int x1, int y1, int x2, int y2) {
            Vertex v=get(x1, y1);
            if (v==null) return false;
            return v.deleteNeighbour(x2, y2);
      }

      /**
       * Finds vertex at specified coordinates and returns its degree (number
of
       * edges that connect to it).
       * @param x X coordinate of the vertex.
       * @param y Y coordinate of the vertex.
       * @pre class invariant of <code>this</code> is true
       * @post <code>this</code> remains unchanged
       * @return degree of the vertex or -1 if the graph does not contain
vertex
       *          at specified coordinates
       * @author Jevgenijs Jonass.
       * @since 25.02.2009
       */
      public int degree(int x, int y) {
            Vertex v=get(x, y);
            if (v==null) return -1;
            return v.degree();
      }

      /**
       * Finds vertex at specified coordinates and returns iterator through
       * its neighbours.
       * The iterator does not allow structural modification of the vertex
set,
       * but you can call <code>setLocation</code> method for any vertex.
       * If you add/delete vertices or edges from the graph while iterating
       * through the vertices, the behaviour of the iterator methods is
undefined
       * (it is not guaranteed that they will raise an exception).
       *
       * <p>The returned VertexIterator will return pointers to Vertex objects
       * that belong to this.vertices (and not to their copies).</p>
       *
       * @param x X coordinate of the vertex.
       * @param y Y coordinate of the vertex.
       * @pre class invariant of <code>this</code> is true
       * @post <code>this</code> remains unchanged
```

```
	 * @return Iterator or null if vertex at given coordinates does not
exist.
	 *          (ownership: caller)
	 * @author Jevgenijs Jonass.
	 * @since 12.03.2009
	 */
	public VertexIterator neighbours(int x, int y) {
		Vertex v=get(x, y);
		if (v==null) return null;
		return v.neighbours();
	}

	/**
	 * Finds vertex at specified coordinates and deletes it from this graph.
	 * If this graph does not contain specified vertex, no changes will be
made.
	 *
	 * @param x X coordinate of the vertex.
	 * @param y Y coordinate of the vertex.
	 * @pre class invariant of <code>this</code> is true
	 * @post class invariant of <code>this</code> is true
	 * @return true if the graph changed as a result of the call.
	 * @author Jevgenijs Jonass.
	 * @since 20.04.2009
	 */
	public boolean deleteVertex(int x, int y) {
		Vertex v=get(x, y);
		if (v==null) return false;
		return v.delete();
	}

	/**
	 * Returns a set containing edges of this graph.
	 *
	 * <p>The returned HashSet will contain pointers to newly allocated
	 * GraphEdge objects, which contain pointers to Vertex objects that
belong
	 * to this.vertices (and not to their copies).</p>
	 *
	 * @pre class invariant of <code>this</code> is true
	 * @post <code>this</code> remains unchanged
	 * @return set containing all edges. Cannot be null. (ownership: caller)
	 * @author Jevgenijs Jonass.
	 * @since 22.04.2009
	 */
	public HashSet <GraphEdge> getEdges() {
		EdgeIterator it=new EdgeIterator(this);
		GraphEdge edge;
		HashSet <GraphEdge> s=new HashSet <GraphEdge> ();
		while (it.hasNext()) {
			edge=it.next();
			s.add(edge);
		}
		return s;
	}

	/**
	 * Returns a set containing vertices of this graph.
	 *
	 * <p>The returned HashSet will contain pointers to Vertex objects that
	 * belong to this.vertices (and not to their copies).</p>
	 *
	 * @pre class invariant of <code>this</code> is true
	 * @post <code>this</code> remains unchanged
```

```
	 * @return set containing all vertices. Cannot be null. (ownership:
caller)
	 * @author Jevgenijs Jonass.
	 * @since 04.05.2009
	 */
	public HashSet <Vertex> getVertices() {
		VertexIterator it=vertices();
		Vertex v;
		HashSet <Vertex> s=new HashSet <Vertex> ();
		while (it.hasNext()) {
			v=it.next();
			s.add(v);
		}
		return s;
	}
}
```

2. Klase Optimization

```
package spw;

import java.awt.*;
import java.util.*;

import graph.*;
import util.RasterImage;
import util.ImageProcessor;
import math.Line;
import math.CoordVector;

/**
 * Class containing methods for image structure analysis.
 * @author Jevgenijs Jonass.
 * @since 07.04.2009
 */
public final class Optimization {

	/**
	 * Connects ends of the edges which are closer than dist.
	 * @param g Graph representing image structure (ownership: caller)
	 * @param dist distance threshold (must be &gt;= 0)
	 * @pre <code>g</code> != null
	 * @pre class invariant of g is true
	 * @post Graph g is modified: all close ends are connected.
	 * @post class invariant of g is true
	 * @author Jevgenijs Jonass.
	 * @since 07.04.2009
	 */
	public static void connectEnds(UGraph g, double dist) {
		assert !(g==null || dist<0);
		HashSet <Vertex> s=new HashSet <Vertex> ();
		VertexIterator it=g.vertices();
		while (it.hasNext()) {
			Vertex v=it.next();
			if (v.degree()==1) s.add(v);
		}
		for (Vertex v: s) {
			for (Vertex w: s) {
				CoordVector a=new CoordVector(v.getX()-w.getX(),
v.getY()-w.getY());
				if (a.getLength()<dist) {
					v.addNeighbour(w);
				}
```

```
                }
            }
        }

        /**
         * Finds key points in the graph.
         * New memory is allocated for the HashSet, but not for objects in it.
         * It means that the returned set contains pointers to vertices that
belong to <code>g</code>.
         *
         * @param g Graph representing image structure (ownership: caller)
         * @param k threshold of the angle cosinus (must be in range [-1..1])
         * @pre g!=null
         * @pre class invariant of g is true
         * @post Graph g is not modified.
         * @return Hash set containing key points (ownership: caller)
         * @author Jevgenijs Jonass.
         * @since 16.04.2009
         */
        public static HashSet <Vertex> findKeyPoints(UGraph g, double k) {
            assert g!=null;
            HashSet <Vertex> keyPoints=new HashSet <Vertex> ();
            VertexIterator it=g.vertices();
            while (it.hasNext()) {
                Vertex v=it.next();
                int deg=v.degree();
                if (deg==1 || deg>2) keyPoints.add(v);
                if (deg==2) {
                    VertexIterator neighbours=v.neighbours();
                    Vertex p=neighbours.next();
                    Vertex q=neighbours.next();
                    double cos=Line.angleBetweenVectorsCos(p, v, q);
                    if (cos>=k) keyPoints.add(v);
                }
            }
            return keyPoints;
        }

        /**
         * Primary optimization of the graph.
         * @param g Graph representing image structure (ownership: caller)
         * @param k threshold of the angle cosinus (must be in range [-1..1])
         * @pre g!=null
         * @pre class invariant of g is true
         * @post class invariant of g is true
         * @author Jevgenijs Jonass.
         * @since 16.04.2009
         */
        public static void primaryOptimization(UGraph g, double k) {
            assert g!=null;
            HashSet <Vertex> deg3=degree3(g);
            Iterator <Vertex> it=deg3.iterator();
            while (it.hasNext()) {
                Vertex v=it.next();
                VertexIterator neighbours=v.neighbours();
                Vertex n1=neighbours.next();
                Vertex n2=neighbours.next();
                Vertex n3=neighbours.next();  assert
neighbours.hasNext()==false;
                final double cos12=Line.angleBetweenVectorsCos(n1, v, n2);
                int f=3;    double minCos=cos12;
                final double cos13=Line.angleBetweenVectorsCos(n1, v, n3);
                if (cos13<minCos) {f=2; minCos=cos13;}
                final double cos23=Line.angleBetweenVectorsCos(n2, v, n3);
```

```
                  if (cos23<minCos) {f=1; minCos=cos23;}
                  //System.out.print("primaryOptimization:");
                  //System.out.print("\nv: " + v + " n1: " + n1 + " n2: " + n2
+ " n3: " + n3);
                  //System.out.println("\ncos12 = " + cos12 + " cos13 = " +
cos13 + " cos23 = " + cos23 + " minCos = " + minCos);
                  if (minCos<k) {
                        switch (f) {
                              case 1: {
                                    Vertex
newVertex=g.addVertex((n2.getX()+n3.getX())/2, (n2.getY()+n3.getY())/2);
                                    if (newVertex!=null) {
                                          v.delete();
                                          newVertex.addNeighbour(n3);
                                          newVertex.addNeighbour(n2);
                                          newVertex.addNeighbour(n1);
                                    }
                                    else {

      //System.out.println("primaryOptimization: replaced vertex is null (NOT
BUG)");
                                    }
                              }
                              break;
                              case 2: {
                                    Vertex
newVertex=g.addVertex((n1.getX()+n3.getX())/2, (n1.getY()+n3.getY())/2);
                                    if (newVertex!=null) {
                                          v.delete();
                                          newVertex.addNeighbour(n3);
                                          newVertex.addNeighbour(n2);
                                          newVertex.addNeighbour(n1);
                                    }
                                    else {

      //System.out.println("primaryOptimization: replaced vertex is null (NOT
BUG)");
                                    }
                              }
                              break;
                              case 3: {
                                    Vertex
newVertex=g.addVertex((n2.getX()+n1.getX())/2, (n2.getY()+n1.getY())/2);
                                    if (newVertex!=null) {
                                          v.delete();
                                          newVertex.addNeighbour(n3);
                                          newVertex.addNeighbour(n2);
                                          newVertex.addNeighbour(n1);
                                    }
                                    else {

      //System.out.println("primaryOptimization: replaced vertex is null (NOT
BUG)");
                                    }
                              }
                              break;
                        }
                  }
                  deg3.remove(v);
                  it=deg3.iterator();
            }
      }

      /**
```

```
       * Edge optimization.
       * @param g graph. Cannot be null. (ownership: caller)
       * @param k variance threshold (must be &gt;=0). Smaller k value means
less changes will be made.
       * @pre class invariant of <code>g</code> is true
       * @post class invariant of <code>g</code> is true
       * @author Jevgenijs Jonass.
       * @since 04.05.2009
       */
      public static void optimizeEdges(UGraph g, double k) {
            assert g!=null && k>=0;
            HashSet <GraphEdge> set=g.getEdges();
            Iterator <GraphEdge> it=set.iterator();
            Vector <Vertex> vect=new Vector <Vertex> ();
            while (it.hasNext()) {
                  vect.clear();
                  GraphEdge ge=it.next();
                  vect.add(ge.first);
                  vect.add(ge.second);
                  do {
                        boolean added1=false;
                        boolean added2=false;
                        if (vect.lastElement().degree()==2) {
                              VertexIterator
vi=vect.lastElement().neighbours();
                              Vertex neigh1, neigh2;
                              neigh1=vi.next(); neigh2=vi.next();
                              if (!vect.contains(neigh1)) {
                                    if (set.contains(new
GraphEdge(vect.lastElement(), neigh1))) {
                                          set.remove(new
GraphEdge(vect.lastElement(), neigh1));
                                          vect.add(neigh1);
                                          added1=true;
                                    }
                              }
                              else {
                                    if (!vect.contains(neigh2)) {
                                          if (set.contains(new
GraphEdge(vect.lastElement(), neigh2))) {
                                                set.remove(new
GraphEdge(vect.lastElement(), neigh2));
                                                vect.add(neigh2);
                                                added1=true;
                                          }
                                    }
                              }
                              if (added1) {
                                    double variance=Line.variance(vect);
                                    //System.out.println("variance: " +
variance);
                                    if (variance>k) {
                                          vect.remove(vect.size()-1);
                                          added1=false;
                                    }
                              }
                        }
                        if (vect.firstElement().degree()==2) {
                              VertexIterator
vi=vect.firstElement().neighbours();
                              Vertex neigh1, neigh2;
                              neigh1=vi.next(); neigh2=vi.next();
                              if (!vect.contains(neigh1)) {
```

```
                                    if (set.contains(new
GraphEdge(vect.firstElement(), neigh1))) {
                                          set.remove(new
GraphEdge(vect.firstElement(), neigh1));
                                          vect.add(0, neigh1);
                                          added2=true;
                                    }
                              }
                              else {
                                    if (!vect.contains(neigh2)) {
                                          if (set.contains(new
GraphEdge(vect.firstElement(), neigh2))) {
                                                set.remove(new
GraphEdge(vect.firstElement(), neigh2));
                                                vect.add(0, neigh2);
                                                added2=true;
                                          }
                                    }
                              }
                              if (added2) {
                                    double variance=Line.variance(vect);
                                    //System.out.println("variance: " +
variance);
                                    if (variance>k) {
                                          vect.remove(0);
                                          added2=false;
                                    }
                              }
                        }
                        if (!added1 && !added2) {
                              //replace vect with one edge
                              Iterator <Vertex> removeItr=vect.iterator();
                              //System.out.println("first: " +
vect.firstElement() + " last: " + vect.lastElement());
                              removeItr.next();
                              Vertex v=removeItr.next();
                              //delete vertices at indexes 1..size-1, where
index starts from 0
                              while (removeItr.hasNext()) {
                                    v.delete();
                                    //System.out.println("delete: "+v);
                                    v=removeItr.next();
                                    assert v!=null;
                              }
                              boolean
b=vect.firstElement().addNeighbour(vect.lastElement());
                              assert vect.size()>1;
                              break;
                        }
                  } while (true);
                  set.remove(new GraphEdge(vect.firstElement(),
vect.lastElement()));
                  it=set.iterator();
            }
      }

      /**
       * Cuts "tails" of the graph, or edge sequences which are shorter than
maxLen.
       * @param g graph. Cannot be null. (ownership: caller)
       * @param maxLen length threshold (must be &gt;=0).
       * @pre class invariant of <code>g</code> is true
       * @post class invariant of <code>g</code> is true
       * @author Jevgenijs Jonass.
```

```
	 * @since 04.05.2009
	 */
	public static void cutTails(UGraph g, double maxLen) {
		assert g!=null && maxLen>=0;
		HashSet <Vertex> s=g.getVertices();
		for (Vertex v: s) if (v.degree()==0) v.delete();
		HashSet <Vertex> K=degree1(g);
		Iterator <Vertex> it=K.iterator();
		Vector <Vertex> L=new Vector <Vertex> ();
		while (it.hasNext()) {
			Vertex V=it.next();
			if (V.degree()!=1) {
				//vertex neighbour may have been deleted during
iteration
				assert V.degree()==0;
				continue;
			}
			L.clear();
			L.add(V);
			Vertex W=V.neighbours().next();
			double len=Line.distance(V, W);
			while (len<maxLen) {
				L.add(W);
				if (W.degree()!=2) break;
				VertexIterator vi=W.neighbours();
				Vertex N=vi.next();
				if (N.equals(L.elementAt(L.size()-2))) N=vi.next();
				len+=Line.distance(W, N);
			}
			L.removeElementAt(L.size()-1);
			for (Vertex v: L) v.delete();
		}
		s=g.getVertices();
		for (Vertex v: s) if (v.degree()==0) v.delete();
	}

	/**
	 * Returns a set containing vertices of the graph which have degree 1.
	 *
	 * <p>The returned HashSet will contain pointers to Vertex objects that
	 * belong to g.vertices (and not to their copies).</p>
	 *
	 * @param g graph. Cannot be null. (ownership: caller)
	 * @pre class invariant of <code>g</code> is true
	 * @post <code>g</code> remains unchanged
	 * @return set containing all vertices. Cannot be null. (ownership:
caller)
	 * @author Jevgenijs Jonass.
	 * @since 04.05.2009
	 */
	public static HashSet <Vertex> degree1(UGraph g) {
		VertexIterator it=g.vertices();
		Vertex v;
		HashSet <Vertex> s=new HashSet <Vertex> ();
		while (it.hasNext()) {
			v=it.next();
			if (v.degree()==1) s.add(v);
		}
		return s;
	}

	/**
	 * Returns a set containing vertices of the graph which have degree 3.
	 *
```

```
       * <p>The returned HashSet will contain pointers to Vertex objects that
       * belong to g.vertices (and not to their copies).</p>
       *
       * @param g graph. Cannot be null. (ownership: caller)
       * @pre class invariant of <code>g</code> is true
       * @post <code>g</code> remains unchanged
       * @return set containing all vertices. Cannot be null. (ownership:
caller)
       * @author Jevgenijs Jonass.
       * @since 04.05.2009
       */
      public static HashSet <Vertex> degree3(UGraph g) {
            VertexIterator it=g.vertices();
            Vertex v;
            HashSet <Vertex> s=new HashSet <Vertex> ();
            while (it.hasNext()) {
                  v=it.next();
                  if (v.degree()==3) s.add(v);
            }
            return s;
      }
}
```

3. Klase SphericalWave

```
package spw;

import java.util.*;
import java.awt.*;

import util.*;
import alg.*;
import graph.*;

/**
 * Class containing spherical wave algorithm.
 * @author Jevgenijs Jonass.
 * @see <a href="http://ocrai.narod.ru/vectory.html">Применение волнового
алгоритма для нахождения скелета растрового изображения</a>
 * @since 28.03.2009
 */
public final class SphericalWave {

      /**
       * Determines how waves will be generated:
       * <ul><li>1 - using 16 neighbour pixels</li>
       * <li>2 - using 4-connectivity</li>
       * <li>3 - using 8-connectivity</li>
       * </ul>
       * @author Jevgenijs Jonass.
       * @since 29.03.2009
       */
      private static int connectivity;

      /**
       * Paints specified <code>pixels</code> on the given image in specified
color.
       * @param pixels Set of pixels. Cannot be null. (ownership: caller)
       *          No object contained in <code>pixels</code> is null.
       *          (ownership of every object in <code>pixels</code>: caller)
       * @param rImage Image to paint on. Cannot be null. (ownership: caller)
       * @param color Pixel color. The highest byte must be 0.
       * @pre class invariant of rImage is true
```

```
	 * @pre coordinates of every Point in <code>pixels</code> are in image
bounds
	 * @post class invariant of rImage is true
	 * @post <code>pixels</code> or any Point object in it will not be
modified
	 * @author Jevgenijs Jonass.
	 * @since 29.03.2009
	 */
	public static void paintPixels(PointSet pixels, RasterImage rImage, int
color) {
		assert !(pixels==null || rImage==null || (color & 0xff000000)!=0);
		int width=rImage.width, height=rImage.height;
		Iterator <Point> it=pixels.iterator();
		while (it.hasNext()) {
			Point i=it.next();
			assert (i.x>=0  && i.x<width && i.y>=0 && i.y<height);
			rImage.pixels[i.y*width+i.x]=color;
		}
	}

	/**
	 * Paints specified <code>pixels</code> on the given image in specified
color.
	 * @param pixels Set of pixels. Cannot be null. (ownership: caller)
	 *          No object contained in <code>pixels</code> is null.
	 *          (ownership of every object in <code>pixels</code>: caller)
	 * @param rImage Image to paint on. Cannot be null. (ownership: caller)
	 * @param color Pixel color. The highest byte must be 0.
	 * @pre class invariant of rImage is true
	 * @pre coordinates of every Point in <code>pixels</code> are in image
bounds
	 * @post class invariant of rImage is true
	 * @post <code>pixels</code> or any Point object in it will not be
modified
	 * @author Jevgenijs Jonass.
	 * @since 29.03.2009
	 */
	public static void paintPixels(HashSet <Vertex> pixels, RasterImage
rImage, int color) {
		assert !(pixels==null || rImage==null || (color & 0xff000000)!=0);
		int width=rImage.width, height=rImage.height;
		Iterator <Vertex> it=pixels.iterator();
		while (it.hasNext()) {
			Vertex i=it.next();
			assert (i.getX()>=0  && i.getX()<width && i.getY()>=0 &&
i.getY()<height);
			rImage.pixels[i.getY()*width+i.getX()]=color;
			//System.out.println("paintPixels: vertex degree=" +
i.degree());
		}
	}

	/**
	 * Calculates average point. Allocates new memory for the resulting
point object.
	 * @param points The points. Cannot be null. (ownership: caller)
	 * @pre points != null and !points.isEmpty()
	 * @post <code>points</code> or any Point object in it will not be
modified
	 * @return Average point. Cannot be null. (ownership: caller)
	 * @author Jevgenijs Jonass.
	 * @since 28.03.2009
	 */
	public static Point averagePoint(PointSet points) {
```

```
            assert !(points==null || points.isEmpty());
            Iterator <Point> it=points.iterator();
            int sumX=0, sumY=0;
            while (it.hasNext()) {
                  Point i=it.next();
                  sumX+=i.x;
                  sumY+=i.y;
            }
            int size=points.size();
            int avgX=sumX/size, avgY=sumY/size;
            return new Point(avgX, avgY);
      }

      /**
       * Paints vertices of the given graph on the image.
       * @param rImage image to paint on. Cannot be null. (ownership: caller)
       * @param g Graph. Cannot be null. (ownership: caller)
       * @pre class invariants of g and rImage are true
       * @pre coordinates of every Vertex in <code>g</code> are in image
bounds
       * @post class invariant of rImage is true
       * @post <code>g</code> will not be modified
       * @author Jevgenijs Jonass.
       * @since 28.03.2009
       */
      public static void drawVertices(RasterImage rImage, UGraph g) {
            assert !(rImage==null || g==null);
            VertexIterator it=g.vertices();
            while (it.hasNext()) {
                  Vertex v=it.next();
                  assert !(v.getX()<0 || v.getX()>=rImage.width || v.getY()<0
|| v.getY()>=rImage.height);
                  rImage.setPixel(v.getX(), v.getY(), 0x00ff0000);
            }
      }

      /**
       * Draws structure of the image. The structure is represented by the
graph.
       * @param rImage image to paint on. Cannot be null. (ownership: caller)
       * @param g Graph. Cannot be null. (ownership: caller)
       * @pre class invariants of g and rImage are true
       * @pre coordinates of every Vertex in <code>g</code> are in image
bounds
       * @post class invariant of rImage is true
       * @post <code>g</code> will not be modified
       * @author Jevgenijs Jonass.
       * @since 30.03.2009
       * @version 19.04.2009 Added support for EdgeIterator.
       */
      public static void drawStructure(RasterImage rImage, UGraph g) {
            assert !(rImage==null || g==null);
            EdgeIterator it=new EdgeIterator(g);
            GraphEdge edge;
            while (it.hasNext()) {
                  edge=it.next();
                  Vertex v1=edge.first, v2=edge.second;
                  assert !(v1.getX()<0 || v1.getX()>=rImage.width ||
v1.getY()<0 || v1.getY()>=rImage.height);
                  assert !(v2.getX()<0 || v2.getX()>=rImage.width ||
v2.getY()<0 || v2.getY()>=rImage.height);
                  Bresenham.line(rImage, v1.getX(), v1.getY(), v2.getX(),
v2.getY(), 0x00ff0000);
            }
```

```
	}

	/**
	 * Finds the connected component in the given area based on 8-
connectivity.
	 * Allocates new memory for the resulting set. Pointers in the resulting
set
	 * can either point to newly created objects or to objects
	 * that belonged to <code>points</code> at the moment before method
call.
	 *
	 * <p>Starts search from the given pixel, adds it to the resulting set
and
	 * removes it from <code>points</code>. After that, processes all its
	 * neighbours that belong to <code>points</code> and so on.</p>
	 *
	 * <p>Finally, all points in the connected component will be removed
from
	 * <code>points</code> and added to the resulting set.</p>
	 *
	 * <p>Does not modify any of the <code>Point</code> objects in
	 * <code>points</code> and does not modify <code>start</code>.</p>
	 *
	 * @param points Determines the search area. Cannot be null. (ownership:
caller)
	 *          No object contained in <code>points</code> is null.
	 *          (ownership of every object in <code>points</code>: caller)
	 * @param start The start pixel. Cannot be null. (ownership: caller)
	 * @pre points.contains(start); points can store pointer to
<code>start</code>
	 *          or to Point object that is equal to <code>start</code>.
	 * @post start will not be modified
	 * @return The connected component. Cannot be null. (ownership: caller)
	 * @author Jevgenijs Jonass.
	 * @since 28.03.2009
	 */
	public static PointSet BFS(PointSet points, Point start) {
		assert !(points==null || start==null || !points.contains(start));
		PointSet connected=new PointSet ();
		Stack <Point> st=new Stack <Point> ();
		st.push(start);
		points.remove(start);
		Point [] a=new Point [8];
		for (int i=0; i<8; i++) a[i]=new Point();
		while (!st.empty()) {
			Point v=st.pop();
			connected.add(v);
			a[0].x=v.x;		a[0].y=v.y-1;
			a[1].x=v.x+1;		a[1].y=v.y-1;
			a[2].x=v.x+1;		a[2].y=v.y;
			a[3].x=v.x+1;		a[3].y=v.y+1;
			a[4].x=v.x;		a[4].y=v.y+1;
			a[5].x=v.x-1;		a[5].y=v.y+1;
			a[6].x=v.x-1;		a[6].y=v.y;
			a[7].x=v.x-1;		a[7].y=v.y-1;
			for (int i=0; i<8; i++) if (points.contains(a[i])) {
				st.push(a[i]);
				points.remove(a[i]);
				a[i]=new Point();
			}
		}
		return connected;
	}
```

```
	/**
	 * Divides set of points into 8-connected components and puts them into
stack.
	 * Processes all <code>points</code> and removes them (and their 8-
connected components)
	 * from the set until it is empty.
	 *
	 * <p>Does not delete or modify existing objects in <code>st</code>.</p>
	 *
	 * <p>New memory for every HashSet object added to <code>st</code> is
allocated.
	 * Every HashSet added to <code>st</code> contains pointers to either
	 * newly created Point objects or objects that belonged to
<code>points</code>
	 * at the moment before method call.</p>
	 *
	 * @param st Stack of wave generations. Cannot be null. (ownership:
caller,
	 *          ownership of all objects in <code>st</code>: caller)
	 * @param points set containing points. Cannot be null. (ownership:
caller)
	 *          No object contained in <code>points</code> is null.
	 *          (ownership of every object in <code>points</code>: caller)
	 * @post all objects added to <code>st</code> are not null.
	 * @post does not modify Point objects which initially belonged to
<code>points</code>
	 * @author Jevgenijs Jonass.
	 * @since 29.03.2009
	 */
	public static void divideWave(Stack <PointSet> st, PointSet points) {
		assert !(st==null || points==null);
		Iterator <Point> it=points.iterator();
		while (it.hasNext()) {
			Point v=it.next();
			PointSet gen=BFS(points, v);
		//	PointSet gen=BFS(points, new Point(v));//for test
			st.add(gen);
			it=points.iterator();
		}
	}

	/**
	 * Generates a spherical wave from every pixel in the given set and
returns
	 * the wave generation as a set of pixels. Note: the result may not be
an 8-connected component.
	 *
	 * <p>Allocates new memory for the resulting set and for all Point
objects in it.</p>
	 *
	 * <p>Only those pixels will be added to the resulting set whose
coordinates
	 * are in image bounds, were not visited (at the start moment) and are
not background.</p>
	 *
	 * <p>Assigns visited[i]=true for all pixels in the resulting set (where
i is coordinate of the pixel).</p>
	 *
	 * @param pixels The initial set of pixels. All pixels must be in image
bounds,
	 *          must be visited and must not be background pixels.
	 *          <code>pixels</code> cannot be null and no object contained
in <code>pixels</code>
	 *          can be null. (ownership of <code>pixels</code>: caller,
```

```
         *          ownership of every object in <code>pixels</code>: caller)
         * @param bgr Determines background pixels.
         *          bgr[i]=true if pixel i is background pixel. Cannot be null.
(ownership: caller)
         * @param visited Determines visited pixels.
         *          visited[i]=true if pixel i was visited. Cannot be null.
(ownership: caller)
         * @param width Image width (must be &gt;0)
         * @param height Image height (must be &gt;0)
         * @pre bgr.length == visited.length == width*height
         * @pre bgr != visited
         * @pre for all i from 0 to bgr.length-1 [visited[i] =&gt; !bgr[i]]
         * @pre start pixel is not background pixel and is not visited
         * @post bgr will not be modified
         * @post start will not be modified
         * @post class invariant of <code>waves</code> is true
         * @post class invariant of <code>g</code> is true
         * @post width and height of <code>waves</code> will not be modified
         * @post Does not modify <code>pixels</code> or any of the
<code>Point</code> objects in <code>pixels</code>.
         * @post does not modify bgr
         * @return The wave generation. Cannot be null. (ownership: caller)
         * @author Jevgenijs Jonass.
         * @since 28.03.2009
         */
        public static PointSet generateWave(
              PointSet pixels, boolean [] bgr, boolean [] visited, int width,
int height) {

              assert !(pixels==null || bgr==null || visited==null ||
bgr==visited || width<1 || height<1 ||
                    bgr.length!=width*height || visited.length!=width*height);

              PointSet newGen=new PointSet ();
              Iterator <Point> it=pixels.iterator();
              Point [] gen;
              switch (connectivity) {
                    case 1: {
                          gen=new Point [16];
                          for (int i=0; i<16; i++) gen[i]=new Point();
                          while (it.hasNext()) {
                                Point i=it.next();
                                assert !(i.x<0 || i.x>=width || i.y<0 ||
i.y>=height ||
                                      !visited[i.y*width+i.x] ||
bgr[i.y*width+i.x]);
                                gen[0].x=i.x;     gen[0].y=i.y-3;
                                gen[1].x=i.x+1; gen[1].y=i.y-3;
                                gen[2].x=i.x+2; gen[2].y=i.y-2;
                                gen[3].x=i.x+3; gen[3].y=i.y-1;
                                gen[4].x=i.x+3; gen[4].y=i.y;
                                gen[5].x=i.x+3; gen[5].y=i.y+1;
                                gen[6].x=i.x+2; gen[6].y=i.y+2;
                                gen[7].x=i.x+1; gen[7].y=i.y+3;
                                gen[8].x=i.x;     gen[8].y=i.y+3;
                                gen[9].x=i.x-1;   gen[9].y=i.y+3;
                                gen[10].x=i.x-2; gen[10].y=i.y+2;
                                gen[11].x=i.x-3; gen[11].y=i.y+1;
                                gen[12].x=i.x-3; gen[12].y=i.y;
                                gen[13].x=i.x-3; gen[13].y=i.y-1;
                                gen[14].x=i.x-2; gen[14].y=i.y-2;
                                gen[15].x=i.x-1; gen[15].y=i.y-3;
                                for (int j=0; j<16; j++) {
                                      int index=gen[j].y*width+gen[j].x;
```

```
                                    if (gen[j].x>=0 && gen[j].x<width &&
gen[j].y>=0 && gen[j].y<height &&
                                    !visited[index] && !bgr[index] &&
!newGen.contains(gen[j])) {
                                          newGen.add(gen[j]);
                                          visited[index]=true;
                                          gen[j]=new Point();
                                    }
                              }
                        }
                  }
                  break;
                  case 2: {
                        gen=new Point [4];
                        for (int i=0; i<4; i++) gen[i]=new Point();
                        while (it.hasNext()) {
                              Point i=it.next();
                              assert !(i.x<0 || i.x>=width || i.y<0 ||
i.y>=height ||
                                    !visited[i.y*width+i.x] ||
bgr[i.y*width+i.x]);
                              gen[0].x=i.x;     gen[0].y=i.y-1;
                              gen[1].x=i.x;     gen[1].y=i.y+1;
                              gen[2].x=i.x-1; gen[2].y=i.y;
                              gen[3].x=i.x+1; gen[3].y=i.y;
                              for (int j=0; j<4; j++) {
                                    int index=gen[j].y*width+gen[j].x;
                                    if (gen[j].x>=0 && gen[j].x<width &&
gen[j].y>=0 && gen[j].y<height &&
                                    !visited[index] && !bgr[index] &&
!newGen.contains(gen[j])) {
                                          newGen.add(gen[j]);
                                          visited[index]=true;
                                          gen[j]=new Point();
                                    }
                              }
                        }
                        connectivity=3;
                  }
                  break;
                  case 3: {
                        gen=new Point [8];
                        for (int i=0; i<8; i++) gen[i]=new Point();
                        while (it.hasNext()) {
                              Point i=it.next();
                              assert !(i.x<0 || i.x>=width || i.y<0 ||
i.y>=height ||
                                    !visited[i.y*width+i.x] ||
bgr[i.y*width+i.x]);
                              gen[0].x=i.x;     gen[0].y=i.y-1;
                              gen[1].x=i.x;     gen[1].y=i.y+1;
                              gen[2].x=i.x-1; gen[2].y=i.y;
                              gen[3].x=i.x+1; gen[3].y=i.y;
                              gen[4].x=i.x+1; gen[4].y=i.y+1;
                              gen[5].x=i.x-1; gen[5].y=i.y+1;
                              gen[6].x=i.x+1; gen[6].y=i.y-1;
                              gen[7].x=i.x-1; gen[7].y=i.y-1;
                              for (int j=0; j<8; j++) {
                                    int index=gen[j].y*width+gen[j].x;
                                    if (gen[j].x>=0 && gen[j].x<width &&
gen[j].y>=0 && gen[j].y<height &&
                                    !visited[index] && !bgr[index] &&
!newGen.contains(gen[j])) {
                                          newGen.add(gen[j]);
```

```
                                        visited[index]=true;
                                        gen[j]=new Point();
                                    }
                                }
                            }
                            connectivity=2;
                        }
                        break;
                    }
                    return newGen;
                }

                /**
                 * Initiates propagation of a spherical wave through binary image
starting
                 * from the specified pixel and adds information about structure of the
                 * image to the graph. If <code>waves</code>!=null, paints all wave
generations
                 * on the <code>waves</code> image.
                 *
                 * <p>Assigns visited[i]=true for all pixels that the wave propagated
through
                 * (where i is the coordinate of the pixel), including the start
pixel.</p>
                 *
                 * @param bgr Determines background pixels.
                 *          bgr[i]=true if pixel i is background pixel. Cannot be null.
(ownership: caller)
                 * @param visited Determines visited pixels.
                 *          visited[i]=true if pixel i was visited. Cannot be null.
(ownership: caller)
                 * @param width Image width (must be &gt;0)
                 * @param height Image height (must be &gt;0)
                 * @param waves Image that all wave generations will be painted on.
                 *          If <code>waves</code> is null, it will be ignored.
(ownership: caller)
                 * @param g Graph representing image structure. Cannot be null.
(ownership: caller)
                 * @param start Point object containing coordinates of start pixel.
Cannot be null. (ownership: caller)
                 * @param n Minimum size of wave generation (must be &gt;0)
                 * @pre <code>waves</code> != null =&gt; class invariant of
<code>waves</code> is true
                 * @pre <code>waves</code> != null =&gt; <code>waves.width == width &&
waves.height == height</code>
                 * @pre class invariant of <code>g</code> is true
                 * @pre bgr.length == visited.length == width*height
                 * @pre bgr != visited
                 * @pre for all i from 0 to bgr.length-1 [visited[i] =&gt; !bgr[i]]
                 * @pre <code>start</code> is in image bounds
                 * @pre start pixel is not background pixel and is not visited
                 * @post bgr will not be modified
                 * @post start will not be modified
                 * @post class invariant of <code>waves</code> is true
                 * @post class invariant of <code>g</code> is true
                 * @post width and height of <code>waves</code> will not be modified
                 * @author Jevgenijs Jonass.
                 * @since 28.03.2009
                 */
                public static void sphericalWave(
                            boolean [] bgr, boolean [] visited, int width, int height,
                            RasterImage waves, UGraph g, Point start, int n) {
```

```
            assert !(bgr==null || visited==null || bgr==visited || width<1 ||
height<1 ||
                  bgr.length!=width*height || bgr.length!=visited.length ||
                  g==null || start==null || n<=0 ||
                  start.x<0 || start.x>=width || start.y<0 || start.y>=height
||
                  bgr[start.y*width+start.x] ||
visited[start.y*width+start.x]);

            boolean drawWaves=waves!=null;
            if (drawWaves) assert waves.width==width && waves.height==height;
            if (drawWaves) {
            //    waves.width=width;
            //    waves.height=height;
            }

            Stack <Pair <Vertex, PointSet>> stack=new Stack <Pair <Vertex,
PointSet>> ();
            Pair <Vertex, PointSet> tmpPair;
            PointSet generation=new PointSet ();
            Stack <PointSet> tmpStack=new Stack <PointSet> ();

            visited[start.y*width+start.x]=true;
            generation.add(start);
            Vertex curVertex;
            if (1>=n) {
                  curVertex=g.get(start.x, start.y);
                  if (curVertex==null) curVertex=g.addVertex(start.x,
start.y);
            }
            else curVertex=null;
            stack.push(new Pair <Vertex, PointSet> (curVertex, generation));

            while (!stack.empty()) {
                  tmpPair=stack.pop();
                  if (drawWaves) paintPixels(tmpPair.second, waves,
ImageProcessor.getRandomColor());
                  generation=generateWave(tmpPair.second, bgr, visited, width,
height);
                  divideWave(tmpStack, generation);
                  while (!tmpStack.isEmpty()) {
                        generation=tmpStack.pop();
                        int size2=generation.size();
                        if (size2>=n) {
                              Point avg=averagePoint(generation);
                              curVertex=g.get(avg.x, avg.y);
                              if (curVertex==null)
curVertex=g.addVertex(avg.x, avg.y);
                              if (tmpPair.first!=null) {
                                    tmpPair.first.addNeighbour(curVertex);
                              }
                              stack.push(new Pair <Vertex, PointSet>
(curVertex, generation));
                        }
                        else stack.push(new Pair <Vertex, PointSet> (null,
generation));
                  }
            }
      }

      /**
       * Generates a spherical wave from each black pixel and builds structure
of
```

```
       * the image. If <code>waves</code>!=null, paints all wave generations
on the <code>waves</code> image.
       *
       * @param bgr Determines background pixels.    bgr[i]=true if pixel i
is background pixel.
       *          Cannot be null. (ownership: caller)
       * @param width Image width (must be &gt;0)
       * @param height Image height (must be &gt;0)
       * @param waves Image that all wave generations will be painted on.
       *          If <code>waves</code> is null, it will be ignored.
(ownership: caller)
       * @param g Graph representing image structure. Initially, graph will be
cleared.
       *          Cannot be null. (ownership: caller)
       * @param n Minimum size of wave generation (must be &gt;0)
       * @param mode Determines how waves will be generated:
       *          <ul><li>1 - using 16 neighbour pixels</li>
       *          <li>2 - combining 4-connectivity and 8-
connectivity</li></ul>
       * @pre <code>waves</code> != null =&gt; class invariant of
<code>waves</code> is true
       * @pre <code>waves</code> != null =&gt; <code>waves.width == width &&
waves.height == height</code>
       * @pre class invariant of <code>g</code> is true
       * @pre bgr.length == width*height
       * @post bgr will not be modified
       * @post class invariant of <code>waves</code> is true
       * @post class invariant of <code>g</code> is true
       * @post width and height of <code>waves</code> will not be modified
       * @author Jevgenijs Jonass.
       * @since 28.03.2009
       */
      public static void buildStructure(boolean [] bgr, int width, int height,
RasterImage waves, UGraph g, int n, int mode) {
            assert !(bgr==null || width<1 || height<1 ||
bgr.length!=width*height || g==null || n<=0);
            assert mode==1 || mode==2;
            switch (mode) {
                  case 1: connectivity=1;
                  break;
                  case 2: connectivity=2;
                  break;
            }

            boolean drawWaves=waves!=null;
            if (drawWaves) assert waves.width==width && waves.height==height;
            if (drawWaves) {
                  Arrays.fill(waves.pixels, 0x00ffffff);
            //    waves.width=width;
            //    waves.height=height;
            }
            int size=bgr.length;
            boolean [] visited=new boolean [size];
      //    Arrays.fill(visited, false);
            g.clear();
            Point p=new Point();
            for (int i=0; i<size; i++) if (!bgr[i] && !visited[i]) {
                  p.x=i%width;
                  p.y=i/width;
                  sphericalWave(bgr, visited, width, height, waves, g, p, n);
            }
      }
}
```

Dokumetārā lapa

Kvalifikācijas darbs „Teksta zīmju attēla uzlabošana” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka kvalifikācijas darbs veikts patstāvīgi un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto kvalifikācijas darbu un atzīstu to par piemērotu/nepiemērotu (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes studiju programmas „Programmēšana un datortīklu administrēšana” kvalifikācijas pārbaudījumu komisijas sēdē.

Darba vadītājs:
(Vadītāja paraksts) _____

Darbs iesniegts Datorikas fakultātē _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Metodiķe: _____.
(Metodiķes paraksts)

Recenzents_____

Darbs aizstāvēts kvalifikācijas pārbaudījumu komisijas sēdē

_____ prot. Nr. _____, vērtējums

_____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)