Универзитет у Београду
Математички факултет

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Саша Вујошевић

Рачунарско препознавање српске ћирилице

магистарски рад

Komisija:
1. Žarko Mijajlović (mentor)
2. Gordana Pavlović-Lažetić (predsednik)
3. Miodrag Živković
4. Zoran Stojiljković

Odbrana: 24.02-2000; 13h.

Pitanja: ćirilica - latinica
2° rečnik/bez rečnika
Koliko je semantika pristuna.
3. Prepoznavanje delova teksta u slikama.

Београд, 1999.

Садржај

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

1. Увод

Развој рачунара довео је до једног парадокса - иако се највећи дио обраде података данас углавном врши на рачунарима, потрошња папира је повећана. Разлог лежи у чињеници да људи још увијек у комуникацији радије користе папир него одговарајућу електронску форму. Нека новија истраживања су показала да долази до замора код читања било ког текста са резолуцијом мањом од 1200×1200 тачака по инчу. На врхунским рачунарима данашњице резолуција није ни приближно толика, тако да ће папир као медијум још опстајати. С друге стране, обрада текста је неупоредиво лакша ако је текст дат у електронској форми. Дакле, пожељно је сваки документ имати у оба облика (или бар оне које треба даље обрађивати). Било који текст-процесор решава проблем штампања текста на папир. Обратан проблем је тежи - како штампани документ превести у одговарајући електронски запис?

У свијету постоје многи програми који решавају овај проблем. Нажалост, углавном препознају разне латиничне фонтове, и, ако уопште и врше лингвистичку корекцију то ријетко раде за српски језик[1].

Овај рад представља један покушај препознавања штампаног ћириличног текста на српском језику, користећи визуелни и лингвистички прилаз.

[1] Поједини програми као нпр. Cript16, FineReader препознају руску ћирилицу, али не и српску, иако разлике нису велике. Такође лингвистичка корекција грешке у тим програмима није за српски језик.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

1.1. Мотивација

Систем за препознавање текста може се употријебити у разне сврхе:

- Као помоћ слијепим људима - код превођења визуелног записа у звучни сигнал,
- У пошти - за аутоматско препознавање адреса,
- У издавачкој дјелатности и библиотекама - прављење репринта, поправљање изгледа разних издања, додавање индекса итд.,
- У банкама - за читање чекова, картица итд.,
- За многе пословне примјене - за читање и препознавање идентификационих кодова, разних формулара итд.,
- У администрацији (катастар, суд, матичар итд.) - за пребацивање разних архива у електронки облик.

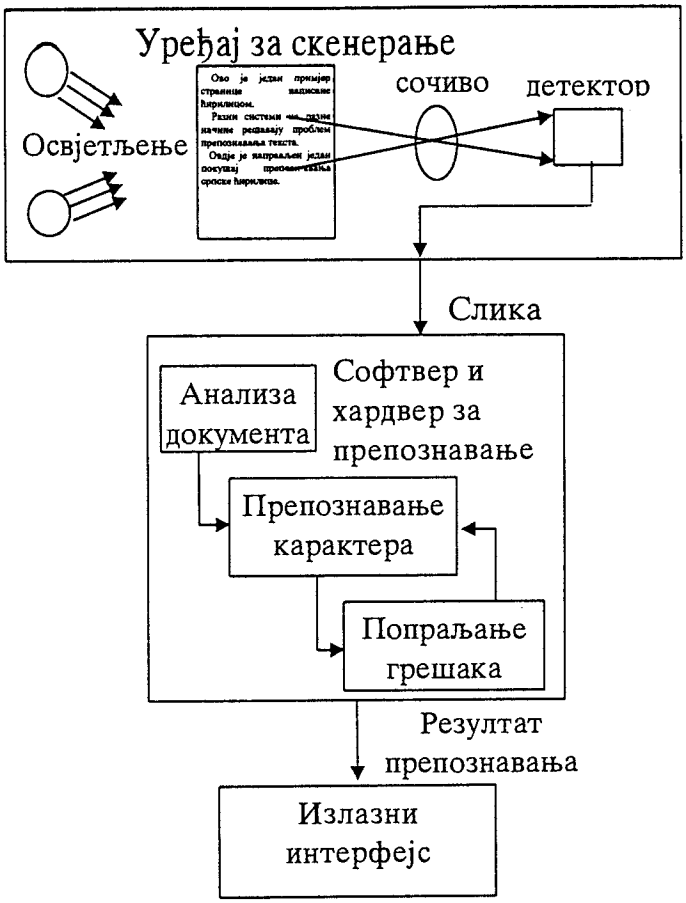
Наравно, зависно од употребе, израда таквог система може бити више или мање сложена. За неке примјене неће бити потребна комплетна азбука, а за неке ће бити потребно препознавање више фонтова (типова слова).

1.2. Систем за препознавање текста

Типичан систем за препознавање текста састоји се из три одвојене логичке компоненте (слика 1.1):

1. уређај за скенирање слика
2. Софтвер и хардвер за препознавање текста
3. Излазни интерфејс

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 1.1: систем за препознавање текста

1. Уређај за скенирање слика. Текст на папиру мора се прво припремити за препознавање. Уређај за скенирање обавља тај задатак - текст на папиру преводи у дигитални запис слике који препознаје рачунар (BMP, TIF, GIF и сл.). Састоји се од четири основне компоненте: детектора (са придруженом електроником), свјетлосног извора, сочива и мјеста за постављање документа. Документ се помоћу свјетлосног извора осветљава и сочиво формира слику документа у детектору. Детектор се састоји од низа елемената од којих сваки конвертује свјетлосни зрак у аналогни сигнал. На крају се аналогни сигнал конвертује у дигиталну слику.

2. Софтвер и хардвер за препознавање текста. Ова компонента је срце система за препознавање текста. Слика преузета од скенера се преводи у низ препознатих слова, који представљају препознати текст.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

У овом раду, бавићемо се углавном овом компонентом, и то њеним софтверским дијелом.

3. Излазни интерфејс. Зависно од даље употребе препознати текст се може различито тумачити. Наиме, питање је да ли ће се тај препознати низ слова користити као чисти текст или је то садржај неких табела или база података итд. Неки комерцијални системи излаз смјештају директно у табеле, базе података или текст процесоре (нпр. WordScan).

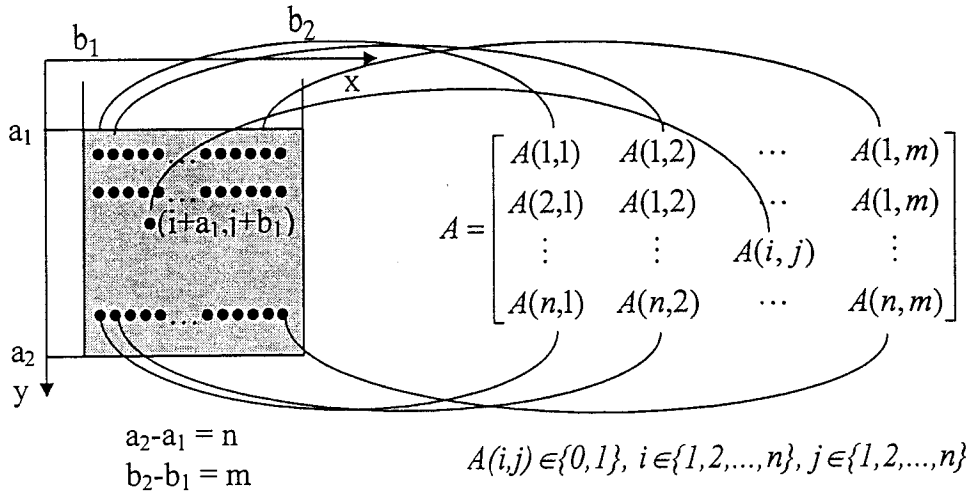
1.3. Терминологија

Морамо правити разлику између текста и слике текста, ријечи и слике ријечи и између слова и слике слова. Ако у овом раду на сваком мјесту то и није наглашено биће јасно из самог контекста.

Ипак, термин графем употребљаваће се за слику слова, а слово (или карактер) за препознату слику слова. Слово је, дакле, апстрактна јединица језика (представљена у рачунару неким ASCII кôдом), а графем његова визуелна презентација (нека бинарна слика, тј. матрица).

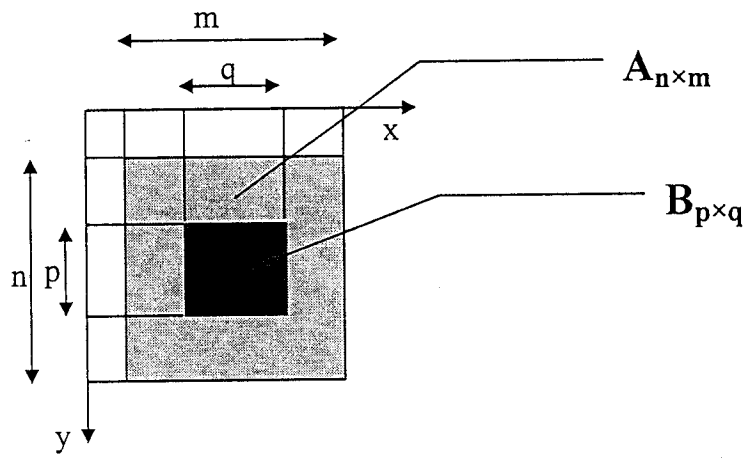
Улазна слика слова, ријечи или текста представља се матрицом. Како се слика састоји од бијелих и црних тачака, то ћемо јој лако придружити бинарну матрицу $A_{n\times m}$, гдје је n висина, а m ширина слике. A(i,j)=1, ако је тачка (i,j) црна, а A(i,j)=0, ако је тачка (i,j) бијела (слика 1.2).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 1.2: представљање слика матрицом

Нека је $p \in \{1,2,...,n\}$ и $q \in \{1,2,...,m\}$. $B_{p\times q}$ је подслика слике $A_{n\times m}$ ако постоје $n_1, n_2 \in \{1,2,...,n\}$ тако да је $B(i,j)=A(i+n_1-1, j+m_1-1)$ за све $i \in \{1,2,...,p\}$, $j \in \{1,2,...,q\}$ (слика 1.2).



Слика 1.3: подслика B слике A

У штампаним документима редови су међусобно паралелни, тако да чак иако је текст приликом скенирања странице ротиран за неки угао, то

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

редови остају паралелни - и они су ротирани за исти угао. Отуда је довољно разматрати подслике типа као на слици 1.3.

У раду су коришћени неки стандардни математички симболи:

$=$ једнакост два објекта.

$\approx$ еквиваленција или сличност два објекта.

$|A|$ број елемената скупа A.

$\varnothing$ празан скуп (или празна листа - у алгоритмима)

1.4. Преглед рада

У раду је дат преглед неких техника за препознавање текста и њихова примјена на ћирилицу и српски језик. Као демонстрација појединих техника настао је програм YuOcr за препознавање српске ћирилице као основног писма.

У глави 2 наведени су неки приступи проблему препознавања текста. У глави 3 дати су алгоритми за испитивање зависности између ријечи. Неки од њх су искоришћени у имплементацији за процес препознавање текста.

Неке могућности корекције грешке је описне су у глави 4, а дате су и табеле најчешћих слова, ријечи и парова слова и ријечи у српском језику. Имплементација је дата у глави 5.

У додацима је дато упутство за употребу апликације YuOcr, као и пар препознатих страница са бројем и процентом препознатих слова и ријечи.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

2. Препознавање текста

Прва фаза у аутоматском препознавању текста је анализа странице. Овдје треба подијелити текст на блокове и линије, а евентуалне слике игнорисати. Затим се текст дијели на ријечи, што због великих празнина између ријечи није тешко извести. За ријеч ћемо, у овој фази, сматрати сваки низ слова између двије празнине (бјелине).

Исправно препознавање текста тако се своди на коректно препознавање ријечи. Сам процес препознавања текста може да дâ више кандидата за сваку издвојену ријеч (тј. за слику те ријечи издвојене из скенираног текста). Одабирање тачно једног кандидата врши се разним визуелним и лингвистичким корекцијама.

2.1. Препознавање текста помоћу препознавања издвојене ријечи

Ријеч је јединица језика у тексту представљена низом слова између два сепаратора (бјелина и знакова интерпункције). Дакле, правићемо незнатну разлику између ријечи настале препознавањем изоловане слике и ријечи као јединице језика (у првом случају се на крају ријечи евентуално налази знак интерпункције). Сваки текст се састоји од одређеног броја ријечи. Питање је како неку слику ријечи препознати, тј. како некој слици придружити одговарајући низ карактера (ниску)?

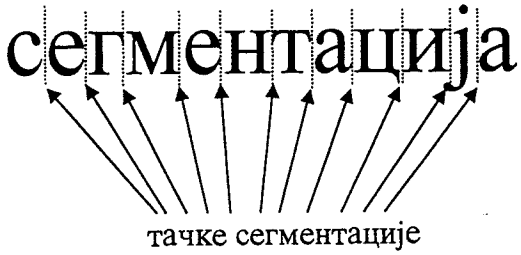
Препознавање издвојене ријечи могуће је извршити на два начина:

- Аналитичким приступом, тј. препознавање слово по слово,
- Холистичким приступом, тј. препознавање читаве ријечи.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

2.1.1. OCR приступ

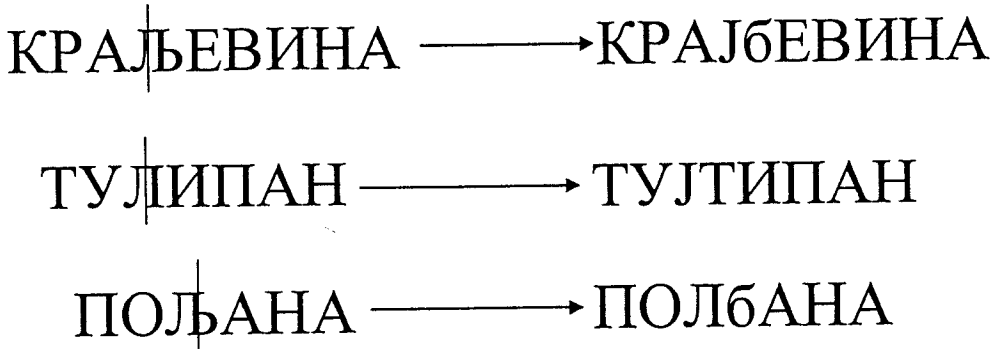
У аналитичком приступу, процес препознавања се састоји из три дијела: сегментације ријечи на слова (слика 2.1), препознавања издвојеног слова и фазе постпроцесирања. Овај приступ се традиционално зове OCR (optical character recognition[2]).



Слика 2.1: сегментација

Кључна компонента, мада не и најтежа, је препознавање слова. Изолованој слици слова треба придружити одговарајући карактер. Многи алгоритми овај проблем решавају мање-више успјешно.

Међутим, иако се препознавање слова врши до на одређену тачност и скоро никад није апсолутно тачно, овај корак ријетко доводи до грешке. Најчешћу грешку, у ствари, ствара сегментација (слика 2.2).

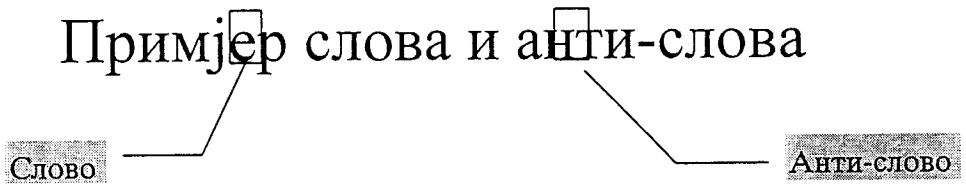


Слика 2.2: примјери погрешне сегментације

[2] Оптичко препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Погрешна сегментација ријечи прави слике које нису слике слова, већ слике тзв. анти-слова. Анти-слово је нека слика настала спајањем дјелова различитих слова (слика 2.3). Јасно је да ће се тада тај низ слова и анти-слова препознати погрешно - правилно издвојена слова вјероватно ће се препознати правилно, али анти-слова скоро сигурно погрешно. Дакле, потребно је обезбиједити добру сегментацију.

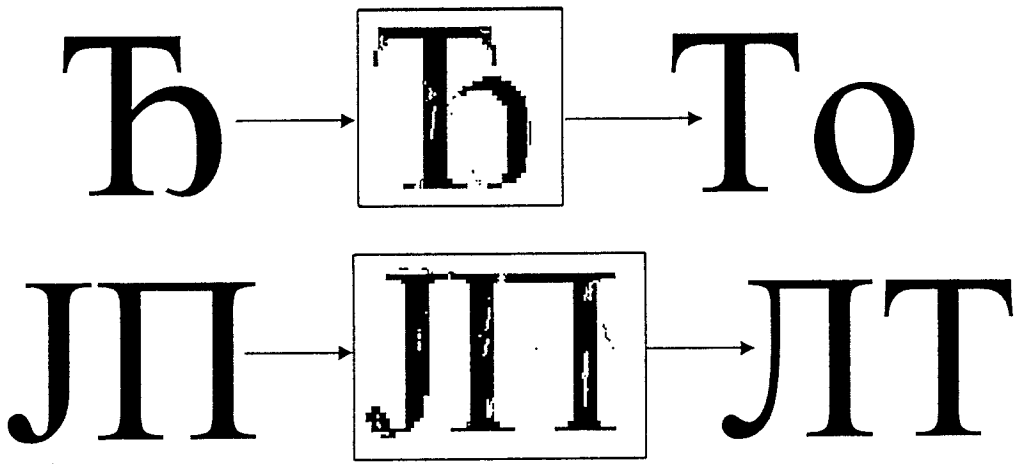


Слика 2.3: примјери слова и анти-слова

Позних 50-их и раних 60-их година препознавање текста се примјењивало на већ подијељен текст на слова. Наиме, штампање (или писање) у разним формуларима обављало се у назначеним пољима исте ширине, па се подјела ријечи на слова свела на читање слова из одговарајућег поља. Овакав начин сегментације мого би се примијенити једино на текстове код којих је свако слово исте ширине, што углавном није случај.

Бољи начин сегментације је тражење празнина између слова. Међутим, ни овај приступ не даје добре резултате, поготово код оштећеног (слабије читљивог) текста. Наиме, слијепљени и разломљени графеми стварају забуну, а ово је још израженије код ћириличних текстова. Зато се један графем може протумачити као два слова, два као једно, а нека група графема као нека друга група слова (слика 2.4).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 2.4: примјери разломљених и слијепљених слова

Ипак, ако би "памтили" групе слова које се додирују или преклапају као један графем поправиле би се и перформансе таквог система.

Разне методе су развијене ради побољшања сегментације. Многе методе почивају на анализирању повезаности компоненти, анализи облика или процјени односа међу карактерима [3,7]. Друге методе "памте" поједине парове слова као један знак [18].

На крају, у фази постпроцесирања, анализом ријечи и анализом слике ријечи, користећи разне лингвистичко-статистичке законитости, врши се корекција грешке. Најједноставнија метода корекције грешке је коришћење великог речника. Идеја је да се препозната ријеч (кандидат) нађе у речнику и тиме потврди тачност ријечи или се не нађе па се предлаже нови визуелно сличан кандидат.

Иако коришћење речника повећава проценат препознатих ријечи, ипак и сам речник може у неким случајевима повећати грешку:

- Ријеч је исправно препозната, али се не налази у речнику, па се даљим постпроцесирањем у ствари прави грешка.
- Ријеч је препозната погрешно, али се та (погрешна) ријеч налази у речнику, па се даља корекција не врши.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

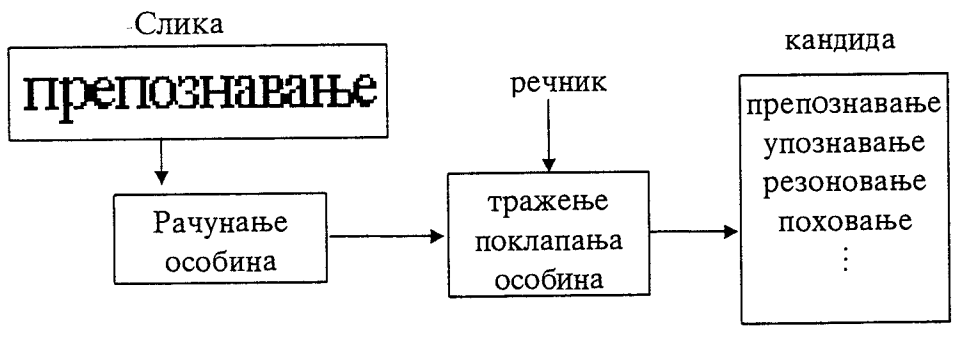
- детектована је грешка, али је за замјену понуђено више сличних (приближно једнако вјероватних) кандидата. Питање је да ли ће бити одабрана права ријеч и да ли је уопште има међу понуђенима.

Речник који садржи све ријечи је тешко направити. Наиме, стално настају нове ријечи, а могуће је и присуство страних ријечи у тексту. Осим тога, у српском језику, постоји и проблем граматике. Велики број падежа, родова и слично, неколико пута би повећали речник или би се у систем морало уградити извођење ријечи користећи граматику српског језика, што је, вјероватно, сложенији проблем од самог препознавања.

О методама корекције грешке биће нешто више ријечи у глави 4.

2.1.2. Холистички приступ

Сличним методама како се препознаје издвојено слово, може се препознати и издвојена ријеч [15]. Треба само у речнику наћи њој визуелно довољно сличну ријеч. На овај начин се сегментација, најтежи дио у OCR приступу, прескаче (слика 2.5).



Слика 2.5: препознавање цијеле ријечи

Иако, на први поглед, овај метод изгледа лакши и бољи од претходног (нема најтежег дијела - сегментације), ипак има један недостатак који га чини чак лошијим од OCR приступа. Наиме, ма колико био велики

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

речник, ипак не садржи све ријечи, па се ријечи ван речника не могу препознати. Даље, свака ријеч се у речник мора смјестити са свим особинама на основу којих се врши препознавање (нпр. слика ријечи) за сваки од фонтова, што додатно оптерећује речник. И на крају, како је већ речено, проблем граматике (падежи, родови, глаголске промјене...) и дијалеката (ијекавица-екавица) у српском језику доводи до новог повећавања речника.

2.2. Препознавање текста помоћу визуелних зависности

Иако препознавање текста помоћу визуелних зависности почива на препознавању ријечи, тај процес није еквивалентан процесу препознавања изоловане ријечи. Користећи визуелне зависности у тексту, дјелови текста (тј. њихове слике - подслике слике читавог текста) се групишу у засебне групе - кластере. Зависно од нивоа разбијања слике почетног текста имамо два основна прилаза: кластеризацију до нивоа ријечи или до нивоа карактера.

У оба случаја идеја је иста - довољно сличне слике, било да су то слике ријечи или слова, групишу се у исти кластер. Сада је довољно препознати једног представника читавог кластера. Такође, могуће је препознати све елементе кластера и упоређивањем резултата извршити корекцију, јер све морају дати исти резултат (било да је то слово или ријеч). Овом методом се прилично успјешно могу препознавати и фонтови.

Уколико текст није превише оштећен, тј. нема превише слијепљених и разломљених слова, то се кластеризација може обавити поклапањем слика (до на довољну тачност). Ако је претходно направљена база примјера (база у којој се чува слика сваког слова азбуке), то се препознавање представника кластера сада се може обавити тражењем сличног поклапања са репрезентима свих слова у бази примјера. Друга метода је метода

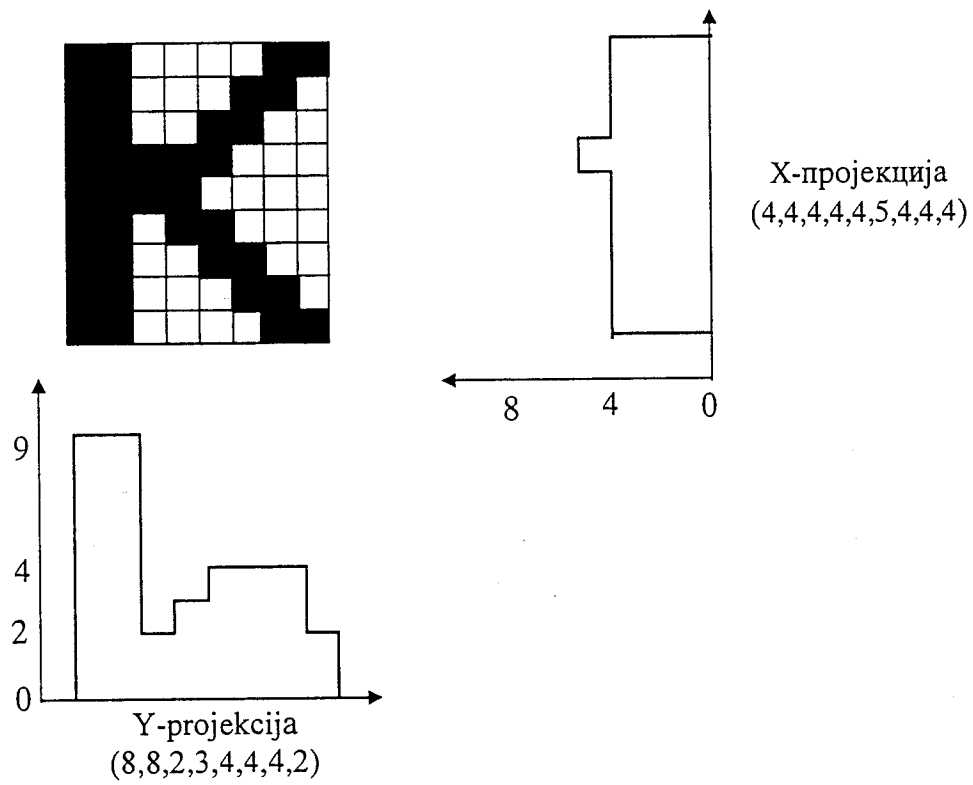
Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

дешифровања. Овдје се у бази не чувају слике слова, већ подаци о разним особинама слова на основу којих се реконструише представник.

На примјер, хоризонтална и вертикална пројекција дају неке карактеристике слова (слика 2.6).

Хоризонтална пројекција (X - пројекција) је вектор чија *i*-та координата показује колико у *i*-тој врсти слике карактера (матрице од нула и јединица) има јединица (црних тачака).

Вертикална пројекција (Y - пројекција) је вектор чија *j*-та координата показује колико у *j*-тој колони слике карактера има јединица.



Слика 2.6: примјери пројекција

Наравно, могуће су и комбинације препознавања путем поклапања представника и дешифровања.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

О препознавању путем дешифровања више се може наћи у [8]. Ипак, пројекција (макар у овом раду) има посебно мјесто, па је стога и издвајамо од осталих. Помоћу пројекције се једноставно и (што је можда и важније) брзо може обавити подјела текста на редове и подјела реда на ријечи. Чак, у неким случајевима, може се извршити и сегментација. Како је празнина између редова (ријечи) велика, то је она отпорна на ротације до пар степени, па се пројекција може употријебити за издвајање редова и ријечи из текста.

2.3. Неке специфичности ћирилице

Свако писмо има своје карактерике које му омогућавају теже или лакше препознавање. На примјер, код кинеског и јапанског писма нема већих проблема са сегментацијом због раздвојених симбола. С друге стране има огроман број приближно сличних симбола, па је ту препознавање симбола главни проблем. Насупрот томе, код арапског писма слова су повезана па је сегментација главни проблем.

Анализирајући разне текстове писане и латиницом и ћирилицом и примјењујући исте тенике препознавања на оба писма, дошли смо до одређених закључака везаних за тополошке карактеристике ћирилице.

Ћирилица и латиница су као писма тополошки слични, тако да се многе технике препознавања латиничног писма могу искористити и за препознавање ћирилице.

И ћирилица и латиница су, за разлику од кинеског и јапанског писма, писма са малим бројем слова. И ћирилица и латиница имају мање-више одвојена слова у ријечима.

Ипак, исти алгоритми не дају исте резултате и за ћирилицу и за латиницу. У ћирилици је велики број слова дио неких других слова (н је садржано у њ, л у љ, ј у л и њ итд.) или им је већи дио једнак (слика 2.7). У латиничном писму је то мање изражено.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Т Ћ Т Ђ Л Љ Н Њ

а)

Ћ Ђ Ц Ц С О

б)

Слика 2.7: а) слова садржана у другим словима; б) слова са истим дјеловима

С друге стране, у ћирилици се само мали број слова продужава испод линије реда (Д, д, ђ, ј, у, ф, Ц, ц, Ц, ц и р). Притом, изузимајући ђ које се може замијенити са ћ, ако се и занемаре тачке испод реда та се слова углавном неће протумачити као нека друга. Стога је одвајање редова нешто лакше у ћириличном писму.

Друга битна карактеристика ћириличног писма је ширина слова. Наиме, у ћириличном писму сва слова су приближно исте ширине, па алгоритми за кластеризацију дају нешто бољи резултат него код латинице. Разлог је због чињенице да су веће слике мање осјетљиве на оштећења. У поглављу 3.2.2 је дат алгоритам за кластеризацију до нивоа карактера и нешто детаљније објашњење предности ширих слова. Нажалост, знаци интерпункције су уски, па кваре перформансе алгоритма.

Даље, сва ћирилична слова, осим ј, су повезана, па се за сегментацију могу искористити и неки алгоритми који почивају на компонентама повезаности. Такође, код ћирилице нема лигатура[3], али има подвлачења слова (енгл. кернинг).

На крају, уз пар изузетака , велика и мала слова у ћириличном писму су визуелно потпуно иста само различите величине. Наиме, само слова а, б, ђ, е, ј, р, с, ћ и ф се разликују од одговарајућих великих слова, а притом с и р небитно. Због тога је база примјера битно мања па се штеди на меморији и,

[3] Пар слова која се у слагању текста слажу као једно. Нпр. у латиничном писму - фи.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

што је битније, добија се на брзини. Истина, на неки начин се мора водити рачуна да ли је у питању велико или мало слово. С друге стране, ова особина отежава препознавање величине слова. Наиме, како су велика и мала слова (углавном) визуелно иста, то је питање да ли је то мало слово одштампано са већом висином или велико слово одштампано са мањом висином.

2.4. О неким програмима за препознавање текста

У оквиру овог рада тестирано је неколико комерцијалних програма за препознавање текста.Неки од њихпрепознају руску дирилицу, неки само латиницу, али врше лингвистичку корекцију за више језика.

Као најбољи програм показао се Acrobat Capture фирме Adobe Systems Inc. Програм препознаје латиницу за више језика - енглески (и то и UK и US), њемачки, француски, холандски, италијански и шведски. Такође има уграђен речник за те језике.

Идеалне текстове препознаје 100%. Такође и текстове са оштећењима, као и ротиране текстове до 3-4 степена препознаје са тачношћу преко 95%. Програм притом успјешно одваја зоне - наслове, пасусе слике итд.

Омогућено је смјештање препознатог текста у фајлове различитог формата, а у једном од њих омогућено је ручно исправљање грешке. Такође препознаје инвертовани текст, што има примјену у издаваштву.

OmniPage Professional Caere Corp., је такође програм за препознавање латинице нешто скромнијих могућности. Има могућност одвајања зона и ручне корекције грешке. Отпоран је на мање ротације - до највише 3 степена. Не види се да ли користи речник.

CuneiForm фирме Cognitive Technology Corp. осим препознавања латинице омогућава и препознавање руске ћирилице. Успјешност препознавања му је мање-више на нивоу претходног програма, стим што има уграђене речнике за руски, енглески, француски и њемачки језик.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

FineReader BIT Software, Inc. је још један програм на нивоу претходних. Препознаје зоне, слике и трпи ротацију текста до 3-4 степена, зависно од квалитета. Препознаје руску ћирилицу, а осим тога и енглеску, њемачку и француску латиницу.

Наведимо на крају и програм Wocar, shareware програм који препознаје енглеску и француску латиницу, као поређење са наведеним комерцијалним програмима. Иако овај програм успјешно препознаје и зоне и текст и слике код идеалних текстова, ипак је веома осјетљив на ротацију и оштећења, па у тим ситуацијама ради врло лоше.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

3. Зависности међу ријечима и њихова примјена

Како је српско писмо алфабетско, то многе ријечи имају заједничких дјелова: неке имају исти почетак, неке исти крај, неке су подријечи других итд. Одговарајуће слике ријечи тада имају сличне зависности.

Свако слово писма може имати више графичких репрезентација - графема. Графеми могу имати разне особине: неке украсе, могу бити подвучени, задебљани или закошени итд. Скуп свих слова азбуке и графема са истим особинама представља фонт.

Визуелне зависности између слика ријечи се могу искористити за један начин кластеризације, за препознавање фонтова, а такође и у фази постпроцесирања.

Кластеризација је груписање сличних објеката. Кластеризација ријечи или слова је, значи, груписање визуелно сличних ријечи или слова у исте групе. Те групе ћемо звати кластери.

Нека су r_1 и r_2 двије ријечи и r_1' и r_2' одговарајуће слике ријечи. Двије ријечи r_1 и r_2 и њима одговарајуће слике r_1' и r_2' могу имати велики број односа. За процес препознавања текста издвојићемо неколико битних:

- r_1' је еквивалентно r_2', у ознаци $r_1' \approx r_2'$. То значи да су слике ријечи r_1' и r_2' довољно сличне, тј. одговарајуће ријечи су исте, тј. $r_1 = r_2$.
- r_1' је подслика слике ријечи r_2', у ознаци $r_1' \approx podslika(r_2')$. Значи, ријеч r_2 је настала конкатенацијом ријечи x, r_1' і y, тј. $r_2 = x \bullet r_1' \bullet y$, гдје симбол • означава конкатенацију. Притом, ријечи x и y могу бити и празне.
- Лијеви дјелови слика r_1' и r_2' су еквивалентни, тј. $prefiks(r_1) = prefiks(r_2)$.
- Десни дјелови слика r_1' и r_2' су еквивалентни, тј. $sufiks(r_1) = sufiks(r_2)$.
- Лијеви дио слике r_1' и десни дио слике r_2' су еквивалентни, тј. $prefiks(r_1) = sufiks(r_2)$.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

- Неки дио слике r_1' еквивалентан је неком дијелу слике r_2', тј. за неке ријечи x_1, y_1, x_2, y_2 и z важи $r_1 = x_1 \bullet z \bullet y_1$ и $r_2 = x_2 \bullet z \bullet y_2$.

Термин "еквивалентан" се примјењује, дакле, за слике које се разликују на невеликом броју тачака, па претпостављамо до су одговарајуће ријечи исте. Истина, овдје се за нека слична слова (нпр. ђ и ћ, џ и ц итд.) може направити грешка, односно неко слово или нека ријеч се може протумачити погрешно. О овом проблему биће нешто више ријечи у глави 3. Примјери зависности међу ријечима дати су на слици 3.1.

Слика 3.1: примјери визуелних релација између слика ријечи

Примијетимо да двије једнаке ријечи не морају имати исте слике, али ће, уколико нема великих оштећења,слике бити еквивалентне.

3.1. Алгоритми за испитивање зависности између ријечи

У овој глави биће наведени алгоритми за испитивање односа између слика ријечи, мада су у имплементацији били коришћени само неки од њих. Сви алгоритми писани су у псеудо паскалу.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Слику ријечи, слова или пак читавог текста лако можемо представити матрицом. Елементи матрице су или 0 или 1, придружени бијелој, односно црној тачки.

3.1.1. Еквиваленција двије слике

Нека су A и B двије $m \times n$ бинарне слике (матрице). Мјера сличности између A и B је

$$s(A,B) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(A(i,j) \wedge B(i,j))}{\sum_{i=1}^{m}\sum_{j=1}^{n}(A(i,j) \vee B(i,j))},$$

гдје су $\wedge$ и $\vee$ логичке операције коњукције и дисјункције (*и* и *или)* редом. Већ је речено да A(i, j), B(i, j) $\in \{0, 1\}$. Нека су $a, b \in \{0, 1\}$. Тада је

$$a \wedge b = ab \text{ и } a \vee b = \begin{cases} 0, a = 0, b = 0 \\ 1, \quad \text{ина ч е} \end{cases}$$

Примијетимо да горња релација нема смисла ако су обије матрице нула матрице. Међутим, како се испитује сличност слика ријечи, то морају постојати црне тачке, тј. одговарајућа матрица није нула матрица.

Очигледно је да је сложеност алгоритма за испитивање еквиваленције двије слике $O(nm)$.

Двије слике су еквивалентне ако је мјера сличности између њих довољно близу 1. Због тога је потребно одредити праг тачности $s_0 \in (0,1)$. Дакле, двије слике A и B су визуелно еквивалентне са прагом тачности s_0 ако је $s(A,B) > s_0$.

Површина слике A је број црних тачака у слици, тј. $|A| = \sum_{m=1}^{M}\sum_{n=1}^{N} A(m,n)$.

Дакле, сличност између двије слике је однос заједничке и укупне површине.

Уколико су слике A и B различитих, али приближно истих димензија тада се мјера сличности дефинише као највеће поклапање између A и B када A прелази преко B.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Могуће је узети нешто другачију мјеру сличности P која се често назива растојање [6], слично Хеминговом растојању[4]. Нека је A бинарна матрица димензија $M \times N$ и B бинарна матрица димензија $K \times L$, које одговарају сликама X и Y редом.

Растојање између X и Y на позицији (p, q) је

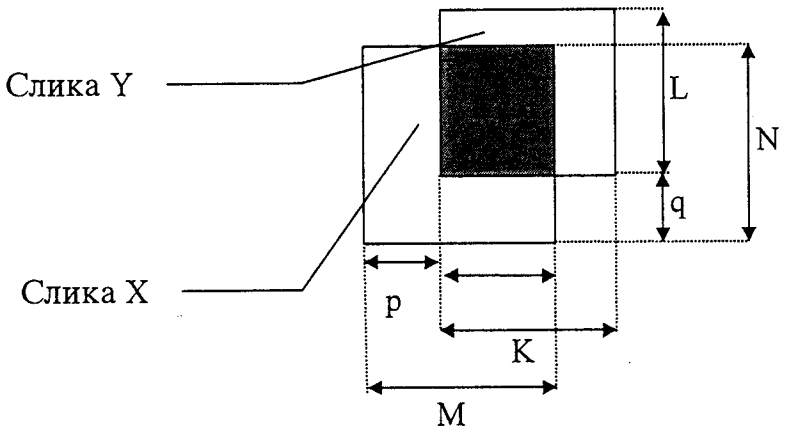
$$R_{XY}(p,q) = \frac{2}{|A|+|B|} \sum_{m=1}^{M} \sum_{n=1}^{N} A(m,n) \wedge B(m-p, n-q).$$

Растојање између X и Y је сада дато са

$$R_{XY} = \max_{p,q} R_{XY}(p,q),$$

по свим могућим p,q (слика 3.2).

Сложеноста алгоритма који на овај начин тражи поклапање слика је $O(n^2m^2)$. Уколико, пак, клизање ограничимо на неколико тачака, тада је сложеност $O(nm)$.



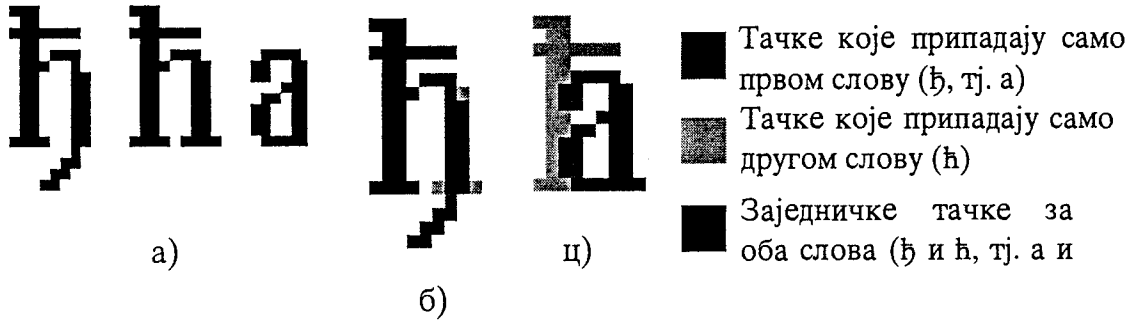
Слика 3.2: Растојање између слика X и Y на позицији (p, q)

Притом, сматраћемо да је $B(a, b) = 0$, ако је $a < 0$ или $b < 0$. Дакле, довољно је посматрати заједничку површину (затамњену на слици) и ту пребројити заједничке црне тачке. Растојање се, значи, тражи тражењем најбољег поклапања "клизајући" једном сликом преко друге.

[4] Ово растојање није еквивалентно растојању у математичком смислу, тј. не задоволјава аксиоме растојања у метричким просторима. Нпр. $d(A,A) \neq 0$.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

На слици 3.3 дата је позиција и мјера сличности између слова ђ, ћ и а. Видимо да се другим приступом добија већа сличност између слика, па ако се узима та мјера сличности, треба повећати праг тачности.



Слика 3.3: Сличност између слова.
а) оригиналне слике слова. б) поклапање слова ђ и ћ је 0,85 (тј. 0.92).
ц) поклапање слова ћ и а је 0.41 (0.58).

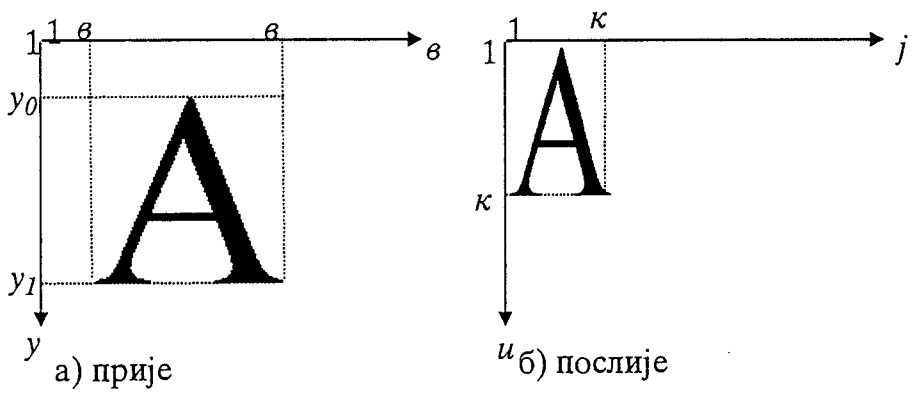
Иако је сличност између слова ђ и ћ прилично велика, тј. могућее већа од задатог прага тачности, уколико нема оштећења у тексту та два слова неће бити замијењена. Наиме, за препознато слово не треба узимати прво које премаши праг тачности, већ оно које има најбоље поклапање. Разложно је претпоставити да ће то бити управо тражено слово. Ако оштећења постоје (неке тачке су избрисане, а неке додате) могуће је да се слово погрешно препозна.

3.1.2. Нормализација

Упоређивање изолованог графема са графемима из базе примјера врши се тражењем најбољег поклапања између слика. Међутим, поклапање је могуће вршити једино уколико су графеми истих (или бар приближно истих) димензија. Отуда се графеми који се упоређују морају довести на исте димензије.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Процес довођења графема на одређене димензије је нормализација. Процес нормализације се спроводи једноставним трансформацијама скалирања (сажимања) и транслације (помјерање) (слика 3.4).



Слика 3.4: Нормализација

Нека су (u_0, v_0) и (u_1, v_1) координате горњег лијевог и доњег десног угла изолованог графема. Нормализацијом на димензије $k_0 \times k_1$ нормализованом графему је (1,1) горни лијеви, а (k_0, k_1) доњи десни угао (слика 3.4).

Послије нормализације имамо да је

$$i=1+k_0(u-u_0)/(u_1-u_0) \text{ и } j=1+k_1(v-v_0)/(v_1-v_0).$$

Дакле, ако је тачка (u, v) била црна, таква ће бити и тачка (i, j) у нормализованој слици.

Према томе, у процесу препознавања изолованог графема, изоловани графем се прво нормализује на димензије графема у бази примјера, па се тек потом врши упоређивање.

Алгоритам нормализације је такође линеаран по броју тачака слике ($O(nm)$).

3.1.3. Кластеризација

Кластеризација је груписање сличних објеката. Кластеризација ријечи или слова је, према томе, груписање визуелно сличних ријечи или

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

слова у исте групе (кластере). У глави 2.2 је већ речено да се кластеризација можи вршити до нивоа ријечи или до нивоа слова.

На слици 3.5 дат је алгоритам за кластеризацију ријечи. Алгоритам је преузет из [13], као и сви алгоритми из ове главе. Такође, алгоритам се може наћи и у [15].

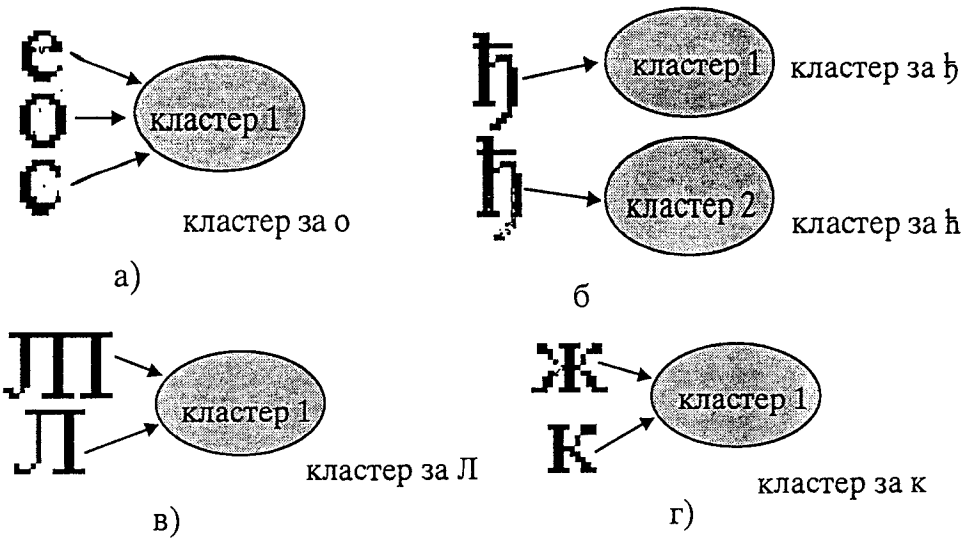
Сложеност алгоритма је *O(knm)*, гдје је *k* број ријечи у тексту, а *n* и *m* висину и ширину слике текста, редом.

```
begin
izdvoj sve slike riječi iz slike teksta
S=∅;
smjesti sve slike riječi u S;
ListaKlastera =∅;
while S≠∅ do
   uzmi sliku s iz S;
   dodat=False;
   for sve klastere K iz ListaKlastera do
      if slika s se poklapa sa prototipom klastera K then
         dodaj sliku s u klaster K;
         dodat=TRUE;
         break;
      endif
   endfor
   if not dodat then
      napravi novi klaster K1 sa prototipom s;
      dodaj s u K1;
      dodaj K1 u ListaKlastera;
   endif
endwhile
end.
```

Слика 3.5: алгоритам за кластеризацију ријечи.

Сваки кластер је репрезентован својим прототипом - произвољним представником кластера. Прототип је, дакле, произвољно одабрана слика из кластера. Прототипови различитих кластера задовољавају неке од визуелних релација, па се ријечи могу даље дијелити и образовати нови кластери. Такође, умјесто препознавања свих ријечи довољно је препознати прототип. Ипак, и овдје је могуће направити грешку. Сличне ријечи (или слова) могу се смјестити у исти кластер иако нису исти, иста се слова могу смјестити у различите кластере итд. (слика 3.6).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 3.6: Примјери погрешне кластеризације:
а) различита слова се смјештају у исти кластер; б) иста слова се смјештају у различите кластере; в) спојена слова се погрешно групишу; г) разломљена слова се погрешно групишу.

Кластеризацијом се може извршити и један вид корекције грешке о чему ће нешто више ријечи бити у глави 4.

3.1.4. Тражење подријечи

Потребно је испитати да ли је слика једне ријечи подслика друге. Дакле, ако су X и Y двије слике, и ако је неки дио слике Y довољно сличан слици X (тј. мјера сличности је већа од задатог прага сличности), тада је X подслика слике Y. Срећна околност је што нас не интересује поклапање на било којем дијелу, већ је довољно прелазити са сликом X преко слике Y само слијева на десно. Ово је могуће због тога што у процесу издвајања ријечи из реда све слике ријечи имају исту или барем приближно исту висину - као висина реда.

Алгоритам за тражење подријечи је дат на слици 3.7.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

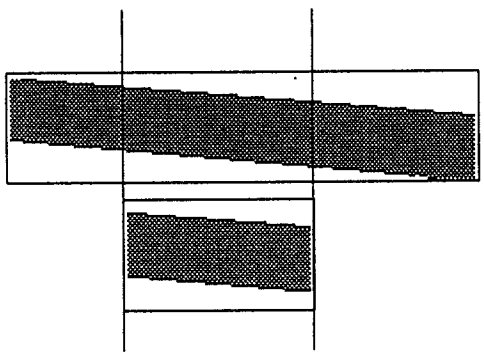
```
function Podslika(Slika1, Slika2, pozicija)

   // Da li je Slika1 podslika Slike2? Vraća TRUE ako jeste,
   // a FALSE ako nije. pozicija pokazuje mjesto poklapanja
   // Slika1: m1×n1; Slika2: m2×n2

if m1>m2 then
   return(FALSE);
else
   for x = 0 to m2 - m1 do
      izdvoj sliku DioSlike2 iz Slika2 od (x,0) do (x+m1,n2);
      if Slika1 ≈ DioSlike2 then
         pozicija = x;
         return(TRUE);
      endif
   endfor
endif
return(FALSE);
end    /* function */
```

Слика 3.7: алгоритам за налажење подслике слике ријечи

Дакле, "клизајући" хоризонтално једном сликом преко друге тражи се поклапање. Већ је речено да и када је текст заротиран редови остају паралелни, па је и тада за издвајање подслика довољно прелазити једном сликом преко друге слијева надесно (слика 3.8). Истина, још боље је ако се истовремено врши и вертикално помјерање за сваку позицију. Како углови ротирања нису превелики, то ово вертикално помјерање неће бити превелико.



Слика 3.8: , "клизање" закошених слика

Ако су димензије слика као у алгоритму, тада је сложеност алгоритма $O(m_1m_2n_1)$. Уколико се врши и вертикално помјерање, сложеност је

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

$O(m_1m_2n_1n_1)$. Као што је речено, вертикално помјерање је довољновршити за невелики број тачака, па можемо сматрати да је и тада сложеност $O(m_1m_2n_1)$.

Питање је да ли се за тражење свих подријечи у читавом тексту мора испитивати однос сваке двије ријечи? Кластеризација нам омогућава да се не испитује однос сваке двије ријечи. Наиме, довољно је испитати однос прототипова свака два различита кластера. Притом, ако се још уреде по ширини прототипа (представника), поправљају се перформансе алгоритма. Алгоритам за означавање парова сликâ ријечи, од којих је једна подслика друге дат је на слици 3.9.

```
begin
Izvrši klasterizaciju slika riječi;
Sortiraj klastere u ListaKlastera u rastućem redosledu po širini prototipa;
while |ListaKlastera|>1 then
   uzmi klaster K iz ListaKlastera;
   for svaki klaster K1 iz ListaKlastera do
      if Podslika(K.prototip, K1.prototip, pozicija) then
         for svaku sliku riječi R iz klastera K do
            for svaku sliku riječi R1 iz klastera K1 do
               označi da je R podslika slike R1;
            endfor
         endfor
      endif
   endfor
endwhile
end.
```

Слика 3.9: алгоритам за означавање парова слика ријечи

Алгоритам се може искористити да би се избјегло непотребно испитивање неких других односа између ријечи. На примјер, ако А није подријеч од В, тада није ни префикс, ни суфикс итд.

Сложеност горњег алгоритма је $O(k^2mn)$, гдје је k број ријечи у тексту, а m и n димензије слике.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

3.1.5. Заједнички почетак ријечи

Алгоритам за испитивање да ли двије ријечи имају исти почетак дат је на слици 3.10. Функција *prefiks* враћа TRUE ако има заједничког почетка, а FALSE ако га нема. Притом промјењива *širina* даје ширину заједничког почетка. Алгоритам користи бинарно претраживање.

```
function prefiks(Slika1, Slika2, širina)

   // Da li Slika1 i Slika2 imaju isti početak
   // širina pokazuje širinu poklapanja
   // Slika1: m1×n1; Slika2: m2×n2

donja = 0;
gornja = min(m1,m2);

while donja < gornja do
   sredina = (donja + gornja)/2;
   izdvoj slike S1 i S2 koje su prefiksi slika Slika1 i Slika2 širine sredina;
   if S1 ≈ S2 then
      donja = sredina;
   else
      gornja = sredina;
   endif
endwhile

širina = donja;
if širina > 0 then
   return(TRUE)
else
   return(FALSE)
endif
end  /* function */
```

Слика 3.10: алгоритам за налажење највећег префикса између двије слике ријечи

Алгоритам за налажење највећег суфикса двије ријечи изгледа скоро исто као за тражење префикса. Сложеност оба алгоритма је $O(m^2n)$. Уколико се еквиваленција између слика обавља клизајућим поклапањем, сложеност је $O(m^2n^2)$, гдје је $m = min\{m_1, m_2\}$ и $n=\{n_1, n_2\}$.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

3.1.6. Заједнички почетак једне ријечи са крајем друге

Испитивање да ли је крај прве ријечи еквивалентан почетку друге обавља функција *prefiks_sufiks* чији је алгоритам дат на слици 3.10. Функција враћа TRUE ако је крај прве слике еквивалентан почетку друге, а FALSE иначе. Притом, ако има поклапања функција даје највеће могуће поклапање и његову ширину преко параметра *širina*.

```
function prefiks_sufiks(Slika1, Slika2, širina)

   // Da li Slika1 ima i isti kraj kao Slika2 početak
   // širina pokazuje širinu poklapanja
   // Slika1: m1×n1; Slika2: m2×n2

m = min(m1,m2);
nađeno = FALSE;
širina1 = m;

while (nađeno = FALSE) and (širina1>0) do
   izdvoj desni dio slike Slika1: S1 širine širina1;
   izdvoj lijevi dio slike Slika2: S2 širine širina1;
   if S1 ≈ S2 then
      nađeno = TRUE;
      širina = širina1;
   else
      širina1 = širina1 - 1;
   endif
endwhile

return(nađeno);

end   /* function */
```

Слика 3.11: Алгоритам за налажење заједничког краја једне и почетка друге ријечи

Сложеност алгоритма је $O(m^2n)$. Уколико се еквиваленција између слика обавља клизајућим поклапањем, сложеност је $O(m^2n^2)$, где је $m = min\{m_1, m_2\}$ и $n=\{n_1, n_2\}$.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

3.1.7. Налажење заједничке подслике

За двије слике слика1 и слика2, алгоритам дат на слици 3.12 испитује да ли имају заједничких подријечи.

```
function zajednicka_slika(Slika1, Slika2)
if postoji slika X tako da
        (podslika(x, Slika1) = TRUE) and
        (podslika(x, Slika2) = TRUE)
then
        return(TRUE)
else
        return(FALSE)
endif
end     /* function */
```

Слика 3.12: Алгоритам за испитивање да ли двије ријечи имају заједничку подријеч

Ако се сматра да су слике исте висине, тада је сложеност алгоритма са слике 3.12 $O(m^2n)$. Уколико нису, па се клизање подслике врши и по вертикали сложеност је $O(m^2n^2)$, гдје је $m = min\{m_1, m_2\}$ и $n=\{n_1, n_2\}$.

3.1.8. Поправљање ефикасности алгоритама

Сви приказани алгоритми су прилично спори јер раде тачку по тачку. Упоређивање двије слике се не може убрзати ако се користи мјера сличности - морају се упоредити све тачке једне слике са свим тачкама друге слике[5]. Уколико би се умјесто мјере сличноси користила нека друга особина која не захтијева коришћење сваке тачке слике, онда би се сложеност еквиваленције двије слике смањила а тиме и свих наведених алгоритама. Наравно, могуће је и комбиновати особине, не идући на поправљање брзине већ ташности препознавања.

Приказани алгоритми се ослањају на тражењу подслика. И овдје се помјерање једне слике преко друге врши тачка по тачка. И ту је могуће направити нека побољшања по угледу на алгоритме за брзо претраживање

[5] У имплементацији је, ипак, направљено мало убрзање техничке природе - користећи рад са битовима умјесто тачка по тачка разматрају се групе по 8 тачака (један бајт)!

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

стрингова (Рабин-Карп - ов алгоритам, Кнут-Морис-Прат - ов алгоритам и Бојер-Мур - ов алгоритам [20]) како је предложено у [13].

Још једна идеја за ефикасније тражење поклапања слика може се извести користећи множење полинома. Наиме, могуће је успоставити везу између множења полинома и свих мјера сличности при клизању једне слике преко друге.

Нека су A и B слике висине 1 и ширине m и n редом. Нека су A и B представљени одговарајућим бинарним матрицама (векторима). Нека су *a, b* и *c* полиноми формирани на следећи начин: $a(x) = a_0 + a_1x + a_2x^2 + a_{m-1}x^{m-1}$, $b(x) = b_0 + b_1x + b_2x^2 + b_{n-1}x^{n-1}$ где је $a_i=A(m-i)$, $i=1,m$; $b_j=B(j)$, $j=1,n$; $c(x)=a(x)b(x)$. Тада коефицијент полинома c_i показује број заједничких тачака слика A и B на *i*-тој позицији.

На сличан начин се могу израчунати и укупне тачке, па према томе и мјера сличности. Ако се рачуна врста по врста, то се идеја може проширити и на слике произвољне висине.

Други пут за побољшање ефикасности је коришћење односа изеђу ријечи на симболичком нивоу. Првобитно препозната страница има приличан број грешака. Ослањајући се на тако препознате ријечи, хипотезе за визуелне односе између ријечи се генеришу преко поклапања стрингова. Поклапање слика се сада користи само за тестирање хипотеза [13].

3.2. Сегментација

Сегментација је подјела текста на слова. Свако изоловано слово се, затим, препознаје на неки начин - користећи разне особине слова. У процесу препознавања текста ово је и најкомплекснији дио, бар када су у питању слабије читљиви текстови.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

3.2.1. Једноставна сегментација

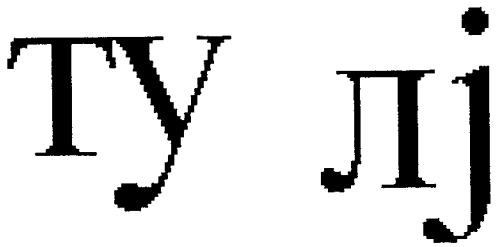
Најједноставнији начин сегментације је тражење празнина између слова. Празнине је најлакше детектовати коришћењем вертикалне пројекције. Наиме, довољно је наћи све вертикалне пројекције слике ријечи. На позицијама на којима су пројекције једнаке 0 (или су довољно мале - ако се зарачуна и нека запрљаност текста) налазе се празнине између слова. Алгоритам за једноставну сегментацију приказан је на слици 3.13.

```
begin
nađi sve vertikalne projekcije riječi R pr(i), i=1,n
označi sve tačke x u kojima je pr(x)=0

for sve tačke x za koje je pr(x-1)≠0 do
   označi tačku x kao tačku segmentacije
endfor
end.
```

Слика 3.13: Алгоритам за једноставну сегментацију

Једноставна сегментација ствара много грешака. Свака два слова која се додирују или преклапају биће погрешно препозната (слика 3.14).



а) б)

Слика 3.14: Карактери који се а) додирују; б) преклапају

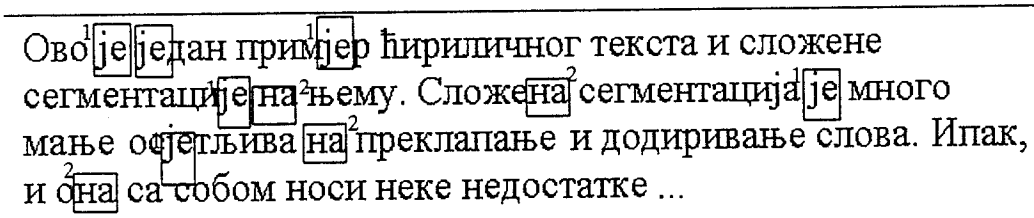
Како је пројекција између слова која се додирују или преклапају ненулта то се та два слова у једноставној сегментацији сматрају за једно, па је јасно да се морају погрешно препознати.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Ипак, могуће је у бази примјера чувати слике свих преклапајућих слова и слике свих парова слова која се додирују. Таквих слова има далеко мање него свих могућих парова па се база примјера неће битно оптеретити, а препознавање се битно поравља.

3.2.2. Сложена сегментација

Користећи зависности између ријечи могуће је извршити један вид сегментације. Идеја је следећа: Кратке ријечи су често садржане у неким дужим па краћа ријеч дијели дужу ријеч на двије или три ријечи (слика 3.15). Настале ниске (или ниска) можда и нису ријечи српског језика, али су вјероватно подријечи неких других дужих ријечи, па се и са њима покушавају подијелити неке ријечи. Поступак се наставља све док има подјела.



Слика 3.15: Примјер текста са односима између ријечи

На слици 3.15 ријечи 1 и 2 (је и на) дијеле неколико других ријечи стварајући нове ниске. Добијају се неке ниске које јесу ријечи српског језика (нпр. дан), али и ниске које то нису (нпр. тљива).

Краћа ниска у дужој, зависно од положаја, прави бар једну нову ниску (слика 3.16).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Слика 3.16: Подјела дуже ријечи помоћу краће ријечи

У примјеру са слике 3.15. ниска *је* дијели ниску *ӣримјер* и одваја слово *р*. Ниска *на* из ниске *она* одваја слово *о*. Настављајући поступак *о* и *р* одвајају нове ријечи и слова. Код дужих текстова ово је још израженије и доводи до одвајања свих (или готово свих) слова. Алгоритам је дат на слици 3.17.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
begin
1. Izdvoj sve slike riječi i smjesti ih u listu W={w1, w2,..., wn}
2. Generiši klastere za sve slike riječi i formiraj listu klastera C={C1, C2,..., Cm}
      gdje je Ci={wi1, wi2,..., wiki}
3. S=∅
4. for sve klastere Ci u C do
5.    Označi proizvoljnu sliku ci iz klastera Ci za prototip klastera Ci
6.    ci.lista_tačaka_segmentacije =∅
7.    S=S ∪ {ci}
8. endfor

9. Q=∅
10. for sve slike ci u S do
11.    kreiraj model p za sliku riječi ci
12.    p.dubina = 0
13.    Q=Q ∪ {p}
14. endfor
       // Podjela prototipa
15. while Q ≠ ∅ do
16.    uzmi model p iz Q
17.    for sve slike ci iz S do
18.       if p je podslika slike ci do
19.          dodaj nove tačke podjele u ci.lista_tačaka_segmentacije
20.          for sve nove fragmente f slike ci do
21.             if f je zadovoljavajuđe veličine then
22.                kreiraj model q za fragment f
23.                q.dubina = p.dubina + 1
24.                Q=Q ∪ {q}
25.             endif
26.          endfor
27.       endif
28.    endfor
29. endwhile
       // Tačke segmentacije za svaku riječ su iste kao kod prototipa
30. for sve slike riječi ci iz S do
31.    for sve slike riječi w iz istog klastera до
32.       pridruži slici w tačke podjele slike si
33.    endfor
34. endfor

35. Prikaži tačke segmentacije za svaku sliku riječi
end.
```

Слика 3.17: Алгоритам за сложену сегментацију

Алгоритам за сложену сегментацију има 5 основних етапа:

1. Издвајање слика ријечи (корак 1)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

2. Кластеризација слика ријечи(корак 2)
3. Генерисање листе прототипова (кораци 3-14)
4. Дијељење слика из листе прототипова (кораци 15-29)
5. Сегментација карактера за све слике ријечи (кораци 30-34)

Нека слика улазног текста димензија $n \times m$. Сложеност етапе 1 је $O(nm)$, а етапе 2 $O(knm)$, гдје је к број ријечи у тексту. Етапе 3 и 5 имају сложеност $O(k)$. Како у листи Q нема више елемената него слова у тексту, то је сложеност етапе 4 $O(lnm^2)$, гдје је l број слова у тексту. Ако се примјењује "клизајуће" поклапање сложеност је $O(ln^2m^2)$. Према томе, сложеност алгоритма је $O(lnm^2)$ (или $O(ln^2m^2)$ за "клизајуће" поклапање).

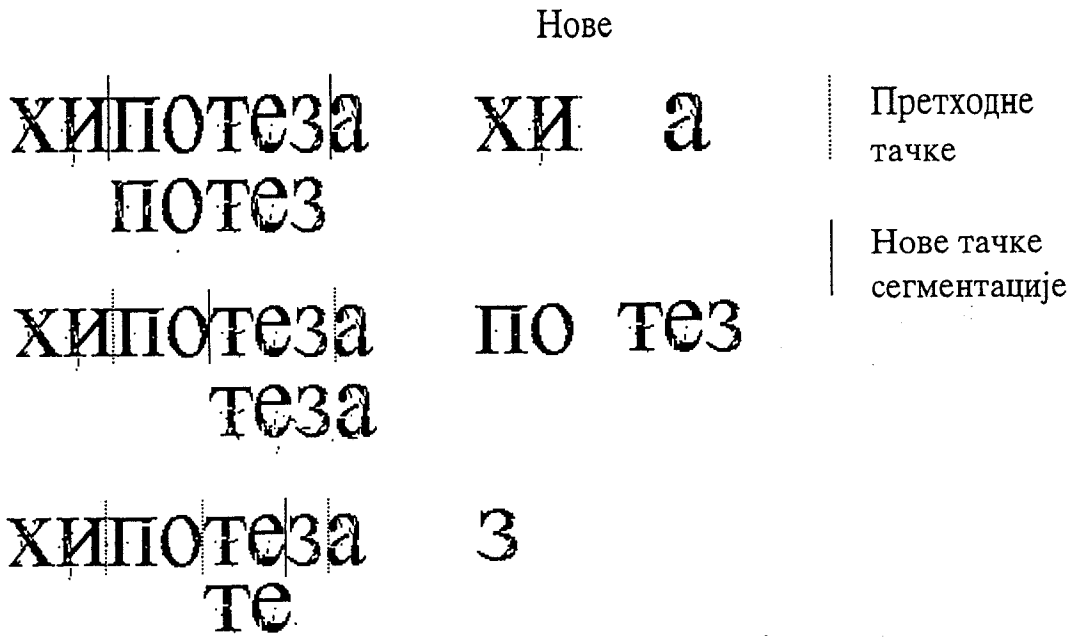
Уколико, као што је предложено у поглављу 3.1.8, користимо множење полинома за ефикасније рачунање мјере сличности то се сложеност може смањити са $O(ln^3)$ на $O(ln^2 logn)$. Ипак, у пракси програм ради брже користећи стандардну мјеру сличности тачку по тачку.

Како су слике у истом кластеру визуелно еквивалентне то све имају исте тачке сегментације. Дакле, ако је прототип коректно подијељен на слова то ће и све ријечи из кластера бити подијељене на исти начин. У алгоритму је листа прототипова означена са S.

Листа (ред) Q садржи све формиране слике - било да су то слике ријечи или слике неког низа слова (који није ријеч српског језика). Поље дубина елемента из Q казује колико је било потребно корака за издвајање те слике. Иницијално, дубина је 0.

Сегментација до нивоа карактера је опсана у корацима 15-29 на слици 3.17. У свакој итерацији из реда Q се узима елемент п и испитује се да ли дијели неку слику ријечи из S. Ако је п подслика слике $c_i \in S$, тада су у c_i одређене двије или једна тачка подјеле. Неки примјери додавања тачака сегментације дати су на слици 3.18.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 3.18: Примјери одређивања тачака сегментације путем поклапања слика

Алгоритам завршава са радом када је Q = ∅. Други начин за прекид рада је ако је достигнута максимална дозвољена дубина.

У корацима 20-26 нови фрагменти се смјештају у ред искључиво ако су довољно широки. Ово из разлога да се ријечи не би дијелиле и преко нивоа слова. На примјер, неко уско слово садржано је у великом броју других слова, па би и ова слова била даље дијељена. На примјер !, (,), итд. су садржане у већини других слова. Тако би заграда подијелила О без потребе. У односу на латиницу, ћирилица је овдје у предности - сва слова (изузимајући специјалне знаке и знаке интерпункције) су довољно широка! Нажалост, с друге стране, многа слова су садржана у другим словима па опет праве погрешну подјелу. На примјер, слово н ће поставити тачку подјеле у сред слова њ и сл.

Такође, приказани алгоритам наслеђује и грешке погрешне кластеризације. Ако су различите ријечи погрешно кластеризоване, оне ће ипак имати исте тачке сегментације што највјероватније није тачно.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Даље, алгоритам ће веома лоше препознавати кратке текстове. Наиме, ако се текст састоји од малог броа ријечи, то је само мали број њих подријеч неких других ријечи у тексту. Штавише, могуће је и да ниједна ријеч није подријеч неке друге. Отуда ће и сегментација бити извршена само до нивоа ријечи или слога, па ће и препознавање бити погрешно.

Из тих разлога, у имплементацији је направљена комбинација једноставне и сложене сегментације. Сâма једноставна сегментација ће извршити дјелимичну подјелу, чиме ће се повећати број кратких ријечи, па ће сложена сегментација дати много боље резултате.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

4. Корекција грешке

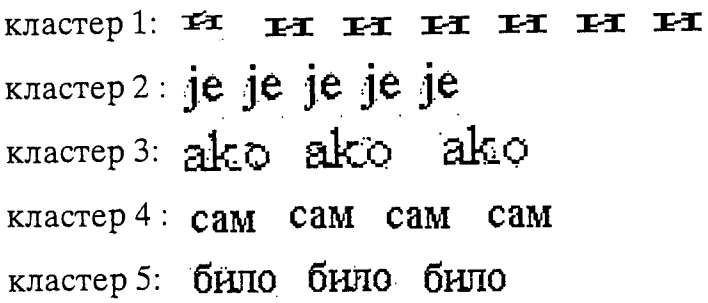
Постпроцесирање (накнадна обрада) је последња фаза у процесу препознавања текста. У овој фази визуелним или лингвистичким приступом врши се корекција препознатог текста. Такође, могућа је и комбинација ова два приступа.

Фаза постпроцесирања, по правилу, знатно повећава број препознатих ријечи. С друге стране, корекција грешке, поготово визуелна, успорава читав процес препознавања текста. Стога треба тражити неке компромисе између тачности и брзине.

4.1. Визуелна корекција грешке

Независно од самог језика и азбуке могуће је извршити корекцију грешке, тзв. визуелну корекцију грешке.

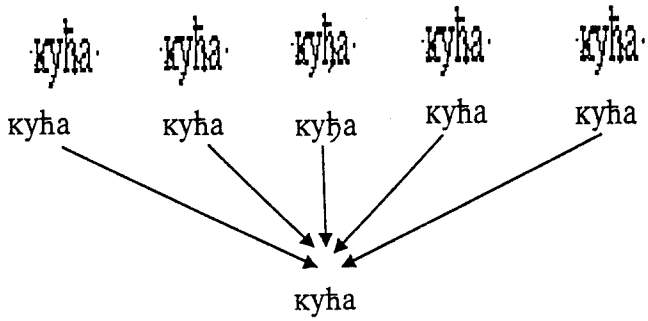
Идеја визуелне корекције грешке почива на кластеризацији до нивоа ријечи. Слика текста је издијељена на слике ријечи, а ове су по сличности груписане у исте или различите кластере (слика 4.1).



Слика 4.1: Примјери кластера за слике ријечи

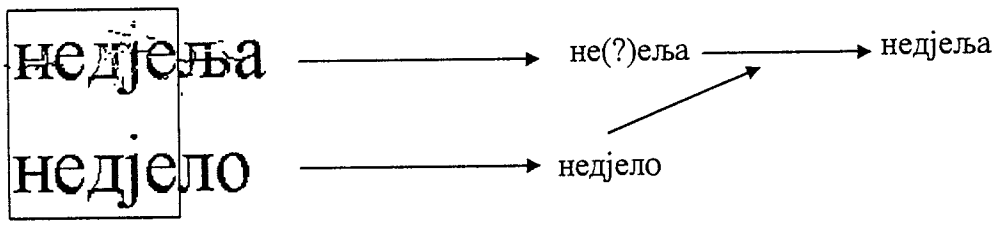
Умјесто препознавања прототипа сваког кластера, препознаје се свака слика ријечи, а затим врши њихово упоређивање. Од свих препознатих ријечи као тачна узима се она која је најчешће препозната (слика 4.2).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 4.2: Бирање најчешће ријечи

Комплекснија анализа слике ријечи доноси још боље препознавање. Ако двије слике ријечи имају заједничку подслику (наравно, до на одређену тачност), тада та подслика мора бити једнако препозната у обије ријечи. Значи, уколико је у једној ријечи тај заједнички дио слабије читљив могуће га је реконструисати преко друге ријечи (слика 4.3). Овај процес је прилично спор јер треба тражити заједничке слике за све парове сликâ ријечи.



Примјер 4.3: Корекција путем визуелних зависности

По правилу свака корекција грешке поправља број препознатих ријечи. Ипак, визуелна корекција грешке наслеђује грешке погрешне кластеризације - нека ријеч иако исправно препозната ће се кориговати ако је смјештена у погрешан кластер. Такође, могуће је да су најчешће препознате ријечи у ствари погрешно препознате, па је корекција погрешна. Штавише, могуће је да је нека од исправно препознатих ријечи тог кластера коригована!

Исте грешке су могуће и ако се врши корекција помоћу визуелних зависности (заједничких подријечи).

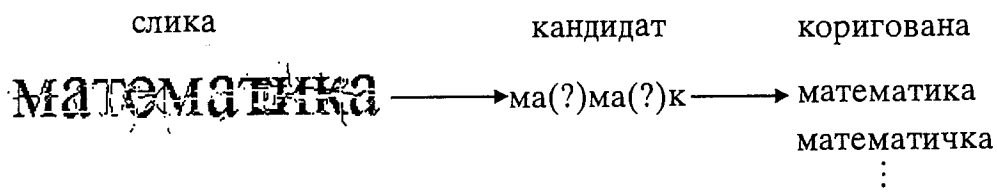
Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

4.2. Лингвистичка корекција

Корекцију грешке је могуће обавити и анализом препознате ријечи. Дакле, овдје се не посматра слика ријечи већ њој придружена ријеч (низ слова).

Типичан приступ лингвистичкој корекцији грешке је коришћење речника. Ријеч за коју се претпоставља да је погрешно препозната треба кориговати. У речнику се тражи ријеч што сличнија препознатој и нађена ријеч се проглашава за коректну. Остаје једино питање како одредити што сличнију ријеч.

У процесу препознавања изолованог слова тражи се мјера сличности између тог слова и одговарајућих примјера из базе. Можемо сва слова која су препозната са задовољавајућом тачношћу означити као тачно препозната, а остала маркирати. Дакле, кориговаће се само ријечи које имају маркираних слова. Притом, у речнику ће се тражити ријеч која се од препознате ријечи разликује само по тим карактерима (слика 4.4).



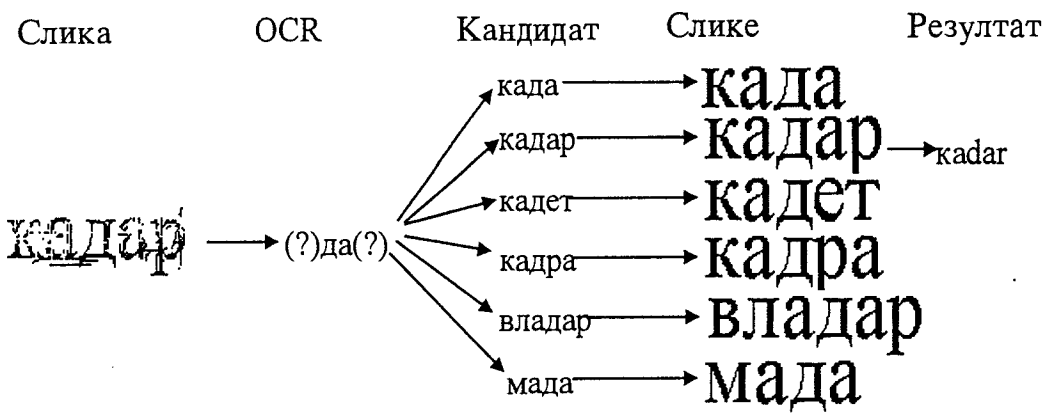
слика 4.4: Корекција ријечи помоћу речника

Коришћење речника поправља перформансе система за препознавање. Ипак, у неким случајевима речник може повећати грешку, како је већ речено у глави 2.

Могуће је направити комбинацију визуелне и лингвистичке корекције. За погрешно препознату ријеч тражи се потенцијални "исправан" кандидат. Затим се прави слика кандидата, па се та слика

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

упоређује са оригиналном сликом ријечи. Од свих кандидата биће одабран онај чија је слика најсличнија оригиналној (слика 4.5).



Слика 4.5: Комбинација визуелних зависности и речника

Већ је речено да речник српског језика мора бити прилично велики или се у систем мора уградити граматика. Дијалекти, падежи, глаголске промјене итд. доводе до тога да је речник превелик. С једне стране то успорава претраживање, а с друге доводи до тога да је немогуће да садржи све ријечи. Уграђивање граматике српског језика у систем је сам по себи комплексан проблем, али би могао смањити речник и побољшати ефикасност.

На крају, споменимо још и комбинацију статистичке зависности ријечи у српском језику и речника.

Прва идеја је да се уз сваку ријеч у речнику памти и учесталост појављивања те ријечи у тексту (тј. у српском језику). Учесталост се добија анализом разних текстова на српском језику. У овом случају је могуће правити специјализоване речнике за разне типове текстова. На примјер, правни речник, математички, економски итд. Јасно је да је у различитим текстовима различита и учесталост појављивања појединих ријечи. Сада се код избора ријечи из речника узима у обзир и њена учесталост у језику. Ако је добијено више кандидата одабраће се онај са највећом учесталошћу. На слици 4.6. приказане су 30 најучесталијих ријечи српског језика[6].

[6] За потребе овог рада урађена је статистика око 15000 различитих ријечи и око 45000 парова ријечи.Укупно је обрађено око 75000 ријечи.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

ријеч	учесталост	ријеч	учесталост	ријеч	учесталост
,	5.599	САМ	0.823	КОЈИ	0.449
.	5.543	ТО	0.645	КАО	0.434
ЈЕ	3.894	ЗА	0.612	КАД	0.419
ДА	3.323	ОНА	0.598	!	0.363
И	2.375	ОД	0.594	СА	0.358
У	2.328	НИЈЕ	0.547	БИЛА	0.357
СЕ	2.262	НЕ	0.543	БИО	0.345
"	1.578	ШТО	0.536	ЈОЈ	0.343
НА	1.279	С	0.527	МИ	0.340
СУ	0.827	БИ	0.460	А	0.325

Слика 4.6: 30 најучесталијих ријечи српског језика са процентом учесталости

Умјесто простог речника могуће је направити речник парова ријечи [13]. Са сваким паром ријечи у речнику памти се и учесталост појављивања тог пара у тексту. Учесталост се, такође, добија анализом разних текстова на српском језику. Опет је могуће правити специјализоване речнике за разне типове текстова. На слици 4.7. приказано је 30 најучесталијих парова ријечи српског језика.

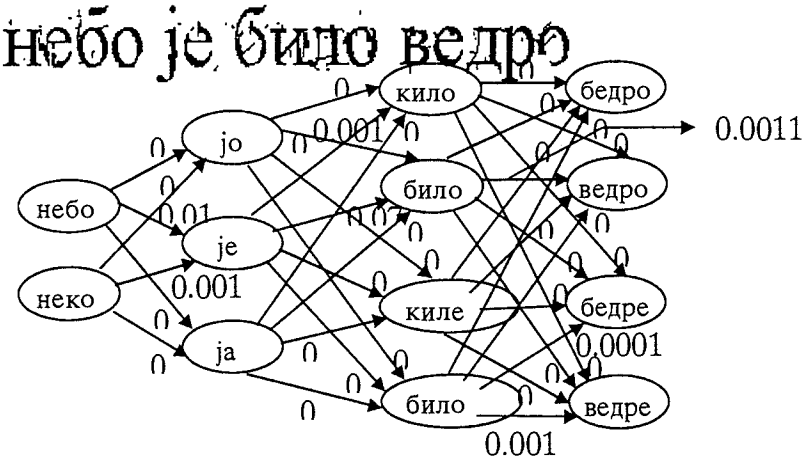
Пар ријечи	учесталост	Пар ријечи	учесталост	Пар ријечи	учесталост
ДА ЈЕ	0.441	. ОНА	0.172	, ДА	0.110
ДА СЕ	0.403	. У	0.164	ДА ЛИ	0.109
" ,	0.360	, РЕЧЕ	0.163	КОЈИ ЈЕ	0.108
. "	0.356	ШТО ЈЕ	0.161	БИО ЈЕ	0.106
, А	0.254	ЈЕ ДА	0.129	ЈЕ БИЛА	0.102
: "	0.223	, У	0.127	. ОН	0.101
ДА БИ	0.194	МИ ЈЕ	0.124	ДА ЋЕ	0.101
" .	0.190	И ДА	0.114	СУ СЕ	0.101
, АЛИ	0.190	, КАО	0.114	ТО ЈЕ	0.099
ЈЕ ТО	0.179	? "	0.113	ЈОЈ ЈЕ	0.099

Слика 4.6: 30 најучесталијих парова ријечи српског језика са процентом учесталости

Свака препозната ријеч се посматра заједно са претходном и следећом ријечи (уколико постоје). Затим се врши корекција тако да тај пар буде што вјероватнији. Идеју је могуће проширити и до нивоа

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

реченице (тј. до неког низа ријечи[7]): прво свакој ријечи одредимо све кандидате из речника који је поправљају. Реченици сада можемо придружити граф: сваком кандидату придружимо чвор и спојимо свака два кандидата који одговарају сусједним ријечима. Свакој грани додијелимо тежину која одговара учесталости тог пара ријечи. Сада је довољно наћи пут максималне тежине и издвојити одговарајућу реченицу (слика 4.8).



Резултат:	nebo je bilo vedro	тежина придруженог пута је 0.0811
Следећи кандидат	nebo je bilo bedro	тежина придруженог пута је 0.08
	nebo je bilo bedre	тежина придруженог пута је 0.08
	nebo je bilo vedre	тежина придруженог пута је 0.08
	neko je bilo vedro	тежина придруженог пута је 0.0721
	⋮	⋮

Слика 4.7: Корекција грешке путем речника и статистике

Уколико неког пара ријечи нема у речнику његова учесталост је 0. Најчешћи парови ријечи имају највећу учесталост.

[7] Тај низ ријечи не мора бити реченица. Штавише, то меже бити низ ријечи из више реченица. Нпр.: ... буде што вјероватнији. Идеју је могуће проширити...

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Примијетимо да је и овим поступком могуће повећати грешку. Да је, на примјер, у речнику пар ријечи < *неко, је* > имао већу учесталост од пара < *небо, је* > реченица би била погрешно препозната[8].

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

[8] За потребе овог рада обрађен је невелики број ријечи, тако да би са комплекснијим речником парова ријечи пар < *неко, је* > заиста имао већу учесталост од пара < *небо, је* >.

5. Имплементација

Програм YuOCR настао је примјеном неких од изложених техника. Програм врши препознавање "чистог" текста (без слика). Такође се врши једноставна корекција грешке помоћу речника.

Програм је писан примјеном програмског пакета Visual C++. У оквиру ове главе биће приложени важнији дјелови кôда, а комплетан кôд је дат у додатку С, као прилог на дискети.

5.1 Имплементација слика

Слика текста, ријечи, реда или, пак, слова у досадашњем тексту је представљана помоћу бинарне матрице. Нажалост, у Visual C++ - у не постоји тип **bool**, тако да је направљена класа *CSlika* за чување сликâ и манипулације са њима.

Декларација класе дата је на слици 5.1.

Атрибути *visina* и *sirina* представљају висину и сирину слике у тачкама. *tacka8* представља 8 тачака. *brBajta* представља број бајтова у реду. Наиме, како не постоји (прави) тип **bool**, а за представљање тачке је довољан један бит, то се у једном бајту (**unsigned char**) памти 8 тачака. Такође, морале су се писати специјалне рутине за приступање појединачној тачки.

Функција **void** *Set*(**int** i, **int** j, **unsigned char** vrijednost = PUNO) на координате (i, j) поставља тражену вриједност (PUNO или PRAZNO). Функција **unsigned char** *Get*(**int** i, **int** j) враћа вриједност са координате (i, j). Функција **int** *Ekviv*(*CSlika** slika2) испитује да ли су двије slike (**this** i slika2) еквивалентне. Враћа проценат сличности између слика.

Функција **int** *SlozenaEkviv*(*CSlika** slika2) ради исто што и претходна функција, стим што тражи максимално поклапање "клизајући" једном преко друге слике.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Функције *zajednickeTackeTip1*, *ukupneTackeTip1*, *zajednickeTackeTip2*, *ukupneTackeTip2* су помоћне функције за лакшу реализацију функције *SlozenaEkviv*. Функција **void** *PodijeliNaRedove*(*NizSlika*& redovi) дијели слику текста на редове. Функција **void** *PodijeliNaRijeci*(*NizSlika*& rijeci, int blanko) дијели ред на ријечи.

Функције

int *Ekvivalentne*(*CSlika** pSlika, *int* nOd),

bool *EkvivalentnaSlika*(CSlika* pSlika, int nTacnost),

int *Podslika*(*CSlika** pSlika, *int* nOd, *int* nPragTacnosti),

int *PostaviTackeSegmentacije*(*int* nOd, *int* nDuzina, *int* nPocetak[], *int* nKraj[]),

int PodslikaPrekoPolinoma(CSlika* pSlika, int nPragTacnosti,
int nPojavljivanja[]);

користе се за сложену сегментацију путем клизања слика, са и без коришћења множења полинома.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
class CSlika
{
protected:
public:
      //Konstruktori
       CSlika();
       CSlika(int sir, int vis);
public:
      // atributi
      int sirina;
      int visina;
      int brBajta;          // Broj bajta u redu
      unsigned char **tacka8; // Nazalost, u VC++ 4.0 nema tipa
          // bool. tacka, u stvari, cuva 8 tacaka slike
         // jedan char = 8 bita
      // funkcije
      ~CSlika();
      int Ekviv(CSlika* slika2);
      int SlozenaEkviv(CSlika* slika2);
      int Ekvivalentne(CSlika* pSlika, int nOd);
      bool EkvivalentnaSlika(CSlika* pSlika, int nTacnost);
      int Podslika(CSlika* pSlika, int nOd, int nPragTacnosti);
      int PostaviTackeSegmentacije(int nOd, int nDuzina,
                        int nPocetak[], int nKraj[]);
      int PodslikaPrekoPolinoma(CSlika* pSlika,
                  int nPragTacnosti, int nPojavljivanja[]);
      int zajednickeTackeTip1(CSlika* slika1, CSlika* slika2,
                              int odV, int odS);
      int ukupneTackeTip1(CSlika* slika1, CSlika* slika2,
                              int odV, int odS);
      int zajednickeTackeTip2(CSlika* slika1, CSlika* slika2,
                              int odV, int odS);
      int ukupneTackeTip2(CSlika* slika1, CSlika* slika2,
                              int odV, int odS);
      void PodijeliNaRedove(NizSlika& redovi);
      void PodijeliNaRijeci( NizSlika& rijeci, int blanko);
      void PodijeliNaSlova( NizSlika& slova);
      CSlika* NapraviPodrijec(int odakle, int dokle);
      CSlika* NapraviPodred(int odakle, int dokle);
      int* NapraviVerProjekciju();
      int* NapraviHorProjekciju();
      void LijevoDesno(int* poc, int* kraj);
      void VrhIDno(int* vrh, int* dno, int* GSuma, int* DSuma);
      CSlika* Normalizovano(int w, int h);
      void Set(int i, int j, unsigned char vrijednost = PUNO );
      unsigned char Get(int i, int j);
};
```

Слика 5.1: Класа CSlika

Функција **void** *PodijeliNaSlova*(*NizSlika*& slova) дијели ријеч на слова.

Функција *CSlika** *NapraviPodrijec*(**int** odakle, **int** dokle) служи за издвајање ријеци из реда. Враћа показивач на нову (издвојену) слику.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Функција CSlika* *NapraviPodred*(int odakle, int dokle) служи за издвајање реда из текста. Враћа показивач на нову (издвојену) слику. И ова и претходна функција су само помоћне функције за реализацију функција *PodijeliNaSlova*, *PodijeliNaRedove* и *PodijeliNaRijeci*.

Функција **int*** *NapraviVerProjekciju*() прави вертикалну пројекцију слике.

Функција **int*** *NapraviHorProjekciju*() прави хоризонталну пројекцију слике.

Функција **void** *LijevoDesno*(**int*** poc, **int*** kraj) тражи стварни почетак и крај слике по ширини.

Функција **void** *VrhIDno*(**int*** vrh, **int*** dno, **int*** GSuma, **int*** DSuma) тражи стварни почетак и крај слике по висини. Успут рачуна и суму вертикалних пројекција при врху и при дну слова.

Функција *CSlika** *Normalizovano*(**int** w, **int** h) је једна од важнијих функција класе *CSlika*. Врши нормализацију слова на димензије h×w.

5.1.1 Дефиниција функција класе *CSlika*

Како је у коментарима самог кôда свака функција објашњена то се нећемо дуже задржавати на објашњењима за сваку функцију. Кôд је дат на слици 5.2.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Konstruktori
CSlika::CSlika()
{
}


CSlika::CSlika(int sir, int vis)
{
  sirina = sir;
  visina = vis;
  tacka8 = new (unsigned char(*[vis]));
  // jedan bajt viska, radi nesto brzih operacija
  brBajta = (sir-1)/8+2;
  for(int i=0; i<vis; i++)
    tacka8[i] = new unsigned char[brBajta];
  // Za svaki red se obezbijedi (oko) 8 puta manje prostora
  // Svaka slika ima dvije tacke segmentacije - pocetak i kraj.
     m_bRijec = false;
     m_lstTackeSegmentacije.AddTail(0);
     m_lstTackeSegmentacije.AddTail(nSirina-1);
}

// Destruktor
CSlika::~CSlika()
{
  for(int i=0; i<visina; i++)
    delete [brBajta] tacka8[i];
  delete[visina] tacka8;
}

// funkcija Ekviv - za karaktere
// Da li su slika1 i slika2 ekvivalentne sa odredjenom tacnoscu
// Podrazumijevaju se potpuno iste dimenzije
// Koristi se iskqucivo poslije normalizacije!!!
int CSlika::Ekviv(CSlika* slika2)//, int tacnost)
{
  int i,j,r,r1;
  r = 0;
  r1 = 0;
  for(i=0; i<visina; i++)
    for(j=0; j<brBajta-1; j++)
      // brzo sabiranje bitova u bajtu x
      for(unsigned char x = tacka8[i][j] |
                            slika2->tacka8[i][j]; x; ++r1)
        x &= x-1; // brzo sabiranje bitova u bajtu
  if(r1==0)
    return 100; // prazno!!!
  for(i=0; i<visina; i++)
    for(j=0; j<brBajta-1; j++)
      // brzo sabiranje bitova u bajtu x
      for(unsigned char x = tacka8[i][j] &
                            slika2->tacka8[i][j]; x; ++r)
        x &= x-1; // brzo sabiranje bitova u bajtu
  return (int) 100*r/r1;
}
```

Слика 5.2: Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// funkcija SlozenaEkviv - za karaktere
// Da li su slika1 i slika2 ekvivalentne sa odredjenom tacnoscu
// Podrazumijevaju se iste dimenzije
int CSlika::SlozenaEkviv(CSlika* slika2)
{
  int slicnost = -1;
  for(int pocI=0; pocI < BALANS; pocI++)
    for(int pocJ = 0; pocJ<BALANS; pocJ++)
    {
      int a1 = ukupneTackeTip1(this, slika2, pocI, pocJ);
      int a2 = ukupneTackeTip1(slika2, this, pocI, pocJ);
      int a3 = ukupneTackeTip2(this, slika2, pocI, pocJ);
      int a4 = ukupneTackeTip2(slika2, this, pocI, pocJ);
      int b1 = zajednickeTackeTip1(this, slika2, pocI, pocJ);
      int b2 = zajednickeTackeTip1(slika2, this, pocI, pocJ);
      int b3 = zajednickeTackeTip2(this, slika2, pocI, pocJ);
      int b4 = zajednickeTackeTip2(slika2, this, pocI, pocJ);
      int slicnost1 = 100*b1/a1;
      int slicnost2 = 100*b2/a2;
      int slicnost3 = 100*b3/a3;
      int slicnost4 = 100*b4/a4;
      int najboqi=
          max(max(slicnost1,slicnost2),max(slicnost1,
slicnost2));
      if(najboqi>slicnost)
        slicnost = najboqi;
    }
  return slicnost;
}
//  Podjela strane na redove
//  vraca broj redova
void CSlika::PodijeliNaRedove(NizSlika& redovi)
{
  // Prvo nadjemo horizontalne projekcije
  int* prH = NapraviHorProjekciju();
  // Zatim nadjemo prvi red, prvu hor. proj razlicitu od 0
  for(int i=0; i<visina; i++)
    if(prH[i]>ZAPRLJANOST_REDA)
    {// Pocinje red - trazimo kraj
      int pocetak = i;
      while(prH[i]>ZAPRLJANOST_REDA && i<visina)
        i++; // Nista - samo se trazi kraj reda!!!
      // Dosli smo do kraja reda (eventualno citave strane)
      int kraj = i-1;
      //visinaReda[brRedova++] = kraj - pocetak + 1;
      redovi.AddTail(NapraviPodred(pocetak, kraj));
    }
  delete [visina]prH;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Jednostavna podjela rijeci na slova
void  CSlika::PodijeliNaSlova( NizSlika& slova)
{
  int* prV = NapraviVerProjekciju();
  // Zatim nadjemo prvu rijec, prvu ver. proj. razlicitu od 0
  for(int i=0; i<sirina; i++)
    if(prV[i]>ZAPRLJANOST_REDA)
    {// Pocinje slovo - trazimo kraj
      int pocetak = i;
      while(prV[++i]>ZAPRLJANOST_REDA && i<sirina)
        ; // Nista - samo se trazi kraj slova!!!
      // Dosli smo do kraja slova (eventualno citave rijeci)
      int kraj = i-1;  // kraj slova
      slova.AddTail(NapraviPodrijec(pocetak, kraj));
    }
  delete [sirina]prV;
};


// Red se dijeli na rijeci
void CSlika::PodijeliNaRijeci(NizSlika& rijeci, int blanko)
{
  // Izmedju dvije rijeci je veliki razmak, bar nekoliko tacaka
  //  ako  ima  zaprljanosti  ona  ima  malo  tacaka  na  citavom
regionu.
  int* prV =  NapraviVerProjekciju();
  int pocetak = -1;
  // Zatim nadjemo prvu rijec, tj. prvu ver. proj.razlicitu od 0
  for(int i=0; i<sirina; i++)
   if(prV[i]>ZAPRLJANOST_REDA)
   {// Pocinje neko slovo, tj. rijec - trazimo kraj
     if(pocetak == -1)
       pocetak = i;
     while(prV[++i]>ZAPRLJANOST_REDA && i<sirina)
       ; // Nista - samo se trazi kraj rijeci!!!
     // Dosli smo do kraja rijeci (eventualno slova)
     int kraj = i-1;  // kraj slova ili rijeci
     while(prV[++i] <= ZAPRLJANOST_REDA && i<sirina)
      ; // Nista - samo se trazi pocetak nove rijeci!!!
     if(i - kraj > blanko || i == sirina)
     {// kraj je kraj rijeci, a ne slova
       rijeci.AddTail(NapraviPodrijec(pocetak, kraj));
     pocetak = -1;
   }
  }
delete [sirina]prV;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Od stare slike pravi novu (podsliku) na datoj poziciji
// po horizontali
// Koristi se za izdvajanje podrijeci iz vece rijeci
// Moze se napraviti da radi brze - preko bitova, slicno
slozenaEkviv
CSlika* CSlika::NapraviPodrijec(int odakle, int dokle)
{
  CSlika* nova;
  nova = new CSlika(dokle-odakle+1, visina);
  for(int i=0; i< nova->visina; i++)
    for(int j=0; j<nova->sirina; j++)
      nova->Set(i, j, Get(i, j+odakle));
  // poslednji se dopunjava sa 0
  for(i=0; i< nova->visina; i++)
    for(int j=nova->sirina; j<(nova->brBajta-1)*8; j++)
      nova->Set(i, j, PRAZNO);
  return nova;
}

// Od stare slike pravi novu (podsliku) na datoj poziciji
// po vertikali
// Koristi se za izdvajanje podreda iz teksta
CSlika* CSlika::NapraviPodred(int odakle, int dokle)
{
  CSlika* nova;
  nova = new CSlika(sirina, dokle-odakle+1);
  for(int i=0; i<nova->visina; i++)
    for(int j=0; j< nova->brBajta-1; j++)
      nova->tacka8[i][j] = tacka8[i+odakle][j]; // Ovo je brze
                                  // nego sa napisanim Get i Set
  // poslednji se dopunjava sa 0
  for(i=0; i<nova->visina; i++)
    for(int j=nova->sirina; j<(nova->brBajta-1)*8; j++)
      nova->Set(i, j, PRAZNO);
  return nova;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Racunanje zajednickih tacaka za dvije slike
// koje se preklapaju tip1              Zajednicki dio
//                                    _____________________/
//                  |Slika1                            /|
//                  |  ,_____________________/  |
//                  | /|//////////////////|   |
//                  |/ |//////////////////|   |
//                  /  |//////////////////|   |
//                 /|__|_____________________|   |
//                /   |              Slika2     |
//         (odS, odV) |_____________________|
//
// Pocetak slike 2 je odV (po visini) i odS (po sirini)
// Pomjeranje slike 2 preko slike 1 je manje od 8 tacaka
// Koristi se poslije normalizacije!!! Istih su dimenzija!
int CSlika::zajednickeTackeTip1(CSlika* slika1, CSlika* slika2,
                                int odV, int odS)
{
  int broj=0;
  for(int i=odV; i<slika1->visina; i++)
   for(int j=0; j<slika1->brBajta-1; j++)
   // ovdje koristimo bajt viska iz inicijalizacije!
     for(unsigned char x = slika2->tacka8[i-odV][j] &
                           ((slika1->tacka8[i][j]<<odS)|
                           (slika1->tacka8[i][j+1]>>(8-odS)));
                          x; ++broj) // brzo sabiranje bitova
       x &= x-1; // brzo sabiranje bitova u bajtu
  return broj;
}
// Racunanje ukupnih tacaka za dvije slike
// koje se preklapaju tip1              Zajednicki dio
//                                    _____________________/_
//                  |Slika1                          /  |
//                  |  ,_____________________/    |
//                  | /|//////////////////|   |
//                  |/ |//////////////////|   |
//                  /  |//////////////////|   |
//                 /|__|_____________________|   |
//                /   |              Slika2     |
//         (odS, odV) |_____________________|
// Pocetak slike 2 je odV (po visini) i odS (po sirini)
// Pomjeranje slike 2 preko slike 1 je manje od 8 tacaka
// Koristi se poslije normalizacije!!! Istih su dimenzija!
int CSlika::ukupneTackeTip1(CSlika* slika1, CSlika* slika2, int
odV, int odS)
{
  int broj=0;
  // Zajednicki dio - tacke i slike1 i slike2
  for(int i=odV; i<slika1->visina; i++)
    for(int j=0; j<slika1->brBajta-1; j++)
    // ovdje koristimo bajt viska iz inicijalizacije!
      for(unsigned char x = slika2->tacka8[i-odV][j] |
                           ((slika1->tacka8[i][j]<<odS) |
                           (slika1->tacka8[i][j+1]>>(8-odS)));
                                  x; ++broj)
       x &= x-1; // brzo sabiranje bitova u bajtu
  return broj;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Racunanje zajednickih tacaka za dvije slike
// koje se preklapaju tip2
//                                        Zajednicki dio
//                                       /
//                        _____________/_
//                       |Slika1       /  |
//                     __|__,_________/    |
//                    | /|/////////////|    |
//                    |/ |/////////////|    |
//                    /  |/////////////|    |
//                   /|  |_____________|____|
//                  / |      Slika2     |
//      (odS, odV)  |___________________|
//
// Pocetak slike 2 je odV (po visini) i odS (po sirini)
// Pomjeranje slike 2 preko slike 1 je manje od 8 tacaka
// Koristi se poslije normalizacije!!! Istih su dimenzija!
int CSlika::zajednickeTackeTip2(CSlika* slika1, CSlika* slika2,
int odV, int odS)
{
  int broj=0;
  for(int i=odV; i<slika1->visina; i++)
    for(int j=0; j<slika1->brBajta-1; j++)
       // ovdje koristimo bajt
       // viska iz inicijalizacije!
    for(unsigned char x = slika2->tacka8[i-odV][j] &
                          ((slika1->tacka8[i][j]<<odS) |
                          (slika1->tacka8[i][j+1]>>(8-odS)));
                                  x; ++broj)
    x &= x-1; // brzo sabiranje bitova u bajtu
  return broj;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Racunanje ukupnih tacaka za dvije slike
// koje se preklapaju tip2
//
//                                        Zajednicki dio
//                          _________________/
//                          |Slika1         /|
//                        __|__,___________/ |
//                        | /|///////////////| |
//                        |/ |///////////////| |
//                        /  |///////////////| |
//                       /|  |_______________|_|
//                      / |      Slika2      |
//       (odS, odV) |____________________|
//
// Pocetak slike 2 je odV (po visini) i odS (po sirini)
// Pomjeranje slike 2 preko slike 1 je manje od 8 tacaka
// Koristi se poslije normalizacije!!! Istih su dimenzija!
int CSlika::ukupneTackeTip2(CSlika* slika1, CSlika* slika2, int
odV, int odS)
{
  int broj=0;
  // Zajednicki dio - tacke i slike1 i slike2
  for(int i=odV; i<slika1->visina; i++)
    for(int j=0; j<slika1->brBajta-1; j++)
     // ovdje koristimo bajt viska iz inicijalizacije!
      for(unsigned char x = slika1->tacka8[i][j] |
                 ((slika2->tacka8[i-odV][j]<<odS) |
                 (slika1->tacka8[i-odV][j+1]>>(8-odS)));
                                    x; ++broj)
        x &= x-1; // brzo sabiranje bitova u bajtu
  return broj;
}

// Normalizacija slike - dovodjenje na dimenzije h x w
// Usput se sracuna stvarna visina i sirina slova
CSlika* CSlika::Normalizovano(int w, int h)
{
  // Prvo se izracuna stvarni pocetak i kraj slova
  int* vPr = NapraviVerProjekciju();
  int* hPr = NapraviHorProjekciju();

  int pocetakVis = -1;
  int krajVis = visina;
  int pocetakSir = -1;
  int krajSir = sirina;

// Vertikalna projekcija
int* CSlika::NapraviVerProjekciju()
{
  // Poslednji bajt je mozda nepotpun...
  int* proj = new int[(brBajta-1)*8];
  // ...Ipak, i za njega racunamo projekcije (bice 0)
  int br = 0;
  for(int i = 0; i<brBajta-1; i++)
    {
    unsigned char maska = 0x80;
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    for(int bit=0; bit<8;bit++)
    {
      int s=0;
      for(int j = 0; j<visina; j++)
        if(tacka8[j][i] & maska)
          s ++;
      maska >>= 1;
      proj[br++] = s;
    }
  }
  return proj;
}
  // Trazenje pocetka i kraja po visini
  while(hPr[++pocetakVis]<=ZAPRLJANOST_REDA)
                // postoji neka tacka8!!!
    ; // Nista - samo se trazi pocetak slike (slova)!!!

  while(hPr[--krajVis]<=ZAPRLJANOST_REDA)
    ; // Nista - samo se trazi kraj slova!!!
  // Trazenje pocetka i kraja po sirini
  while(vPr[++pocetakSir]<=ZAPRLJANOST_REDA)
                                  // postoji neka tacka8!!!
    ; // Nista - samo se trazi pocetak slike (slova)!!!
  while(vPr[--krajSir]<=ZAPRLJANOST_REDA )
    ; // Nista - samo se trazi kraj slova!!!

  // Slovo se nalazi u pravougaoniku
  // (pocetakVis, pocetakSir) do (krajVis, krajSir)

  CSlika* skalirana = new CSlika(w, h);

  for(int i=0; i<h; i++)
    for(int j=0;j<w; j++)
    {
      int StaroI = pocetakVis + (krajVis-pocetakVis+1)*i/h;

      int StaroJ = pocetakSir + (krajSir-pocetakSir+1)*j/w;
      ASSERT(StaroI<=krajVis && StaroI>=pocetakVis);
      ASSERT(StaroJ<=krajSir && StaroJ>=pocetakSir);
      skalirana->Set(i, j, Get(StaroI, StaroJ));
    }
  // dopunjavanje poslednjeg (dodatog) bajta u svakom redu
  for(i=0; i<h; i++)
    skalirana->tacka8[i][skalirana->brBajta-1] = 0x00;
  delete [sirina]vPr;
  delete [visina]hPr;
  return skalirana;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
int* CSlika::NapraviHorProjekciju()
{
  int* proj = new int[visina];
  for(int i = 0; i<visina; i++)
    {
    int s=0;  // zbir jedinica(bitova) u bajtu
    for(int j = 0; j<brBajta-1; j++)
      for(unsigned char x = tacka8[i][j]; x; ++s)
                  // brzo sabiranje
                  //bitova u bajtu
       x &= x-1; // brzo sabiranje bitova u bajtu
    proj[i] = s;
    }
  return proj;
}
// Prva i poslednja tacka8 u slici - po visini
// Takodje racuna sumu horizontalnih projekcija
// za gornju i donju 1/4 slova

void CSlika::VrhIDno(int* vrh, int* dno, int* GSuma, int* DSuma)
{
  // Prvo se izracuna stvarni pocetak i kraj slova
  int* hPr = NapraviHorProjekciju();
  *GSuma = 0;
  *DSuma = 0;
  for(int i=0; i < visina/4; i++)
    *GSuma += hPr[i];
  for(i= visina/4; i < visina; i++)
    *DSuma += hPr[i];
  *vrh = -1;
  *dno = visina;
  // Trazenje pocetka i kraja po visini
  while(hPr[++ *vrh]<=ZAPRLJANOST_REDA)// postoji neka tacka8!!!
    ; // Nista - samo se trazi pocetak slike (slova)!!!
  while(hPr[-- *dno]<=ZAPRLJANOST_REDA)
    ; // Nista - samo se trazi kraj slova!!!
  delete [visina]hPr;
}
void CSlika::LijevoDesno(int* poc, int* kraj)
{
  // Prvo se izracuna stvarni pocetak i kraj slova
  int* vPr = NapraviVerProjekciju();
  *poc = -1;
  *kraj = sirina;

  // Trazenje pocetka i kraja po sirini
  while(vPr[++ *poc]<=ZAPRLJANOST_REDA)// postoji neka tacka8!!!
    ; // Nista - samo se trazi pocetak slike (slova)!!!
  while(vPr[-- *kraj]<=ZAPRLJANOST_REDA)
    ; // Nista - samo se trazi kraj slova!!!
  delete [sirina]vPr;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Na koordinate i,j postavlja tacku - 1 ili 0
// Ako se nista ne naglasi, podrazumijeva se 1 (PUNO)
void CSlika::Set(int i, int j, unsigned char vrijednost /* =
PUNO */)
{
  int bajt = j/8; // Koji bajt?
  int pozicija = j%8; // Pozicija u bajtu
  unsigned char maska = 0x80>>pozicija;
  if(vrijednost)
    tacka8[i][bajt] |= maska;
  else
    tacka8[i][bajt] &= ~maska;
}

// Sa koordinata i,j "cita" tacku - 1 ili 0
unsigned char CSlika::Get(int i, int j)
{
  int bajt = j/8; // Koji bajt?
  int pozicija = j%8; // Pozicija u bajtu
  unsigned char mask = 0x80>>pozicija;
  if(tacka8[i][bajt] & mask)
    return PUNO;
  return PRAZNO;
}

// Da li je tekuca slika (this) podslika slike 'slika'
// pocev od pozicije nOd
// Vraca pocetak prvog pojavljivanja, ako jeste podslika
// -1, ako nije
// Podrazumijeva se ista (priblizno) m_nVisina
int CSlika::Podslika(CSlika* pSlika, int nOd, int nPragTacnosti)
{
     if(m_nSirina + nOd > pSlika->m_nSirina)
          return -1;

     for(int i=nOd; i<pSlika->m_nSirina-m_nSirina + 1; i++)
     {
         int nSlicnost = Ekvivalentne(pSlika, i);
          if(nSlicnost >= nPragTacnosti)
               return i;
     }
     return -1;
}

// Da li je tekuca slika (this) podslika slike 'slika'
// Vraca broj pojavljivanja, ako jeste podslika
// 0, ako nije. Bocno vraca sva pojavljivanja
// Podrazumijeva se ista (priblizno) m_nVisina
// Racuna se preko mnozenja polinoma
int CSlika::PodslikaPrekoPolinoma(CSlika* pSlika,
               int nPragTacnosti, int nPojavljivanja[])
{
     if(m_nSirina > pSlika->m_nSirina)
          return -1;
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
char cPolinom1[1000], cPolinom2[1000],
       cBrojZajednickih[2000], cBrojUkupnih[2000];
int nG[1000], nD[1000];
int i,j;
int nBr = 0;
for(j = 0; j < pSlika->m_nSirina - m_nSirina + 1; j++)
{
      nG[j] += 0;
      nD[j] += 0;
}
for(i = 0; i < pSlika->m_nVisina && i < m_nVisina; i++)
{
      for( j = 0; j < m_nSirina; j++)
            cPolinom1[m_nSirina - 1 - j] = Get(i,j);
            // Unazad
      for(j = m_nSirina; j < pSlika->m_nSirina; j++)
            cPolinom1[j] = 0;
             // Dovodjenje na isti stepen
      for( j = 0; j < pSlika->m_nSirina; j++)
            cPolinom1[j] = pSlika->Get(i,j);
      Mult(pSlika->m_nSirina, cPolinom1,
                cPolinom2, cBrojZajednickih);
      // invertovanje
      for( j = 0; j < m_nSirina; j++)
            cPolinom1[m_nSirina - 1 - j]
                  = cPolinom1[m_nSirina-1-j] ? 0 : 1;
      // Dovodjenje na isti stepen
      for(j = m_nSirina; j < pSlika->m_nSirina; j++)
            cPolinom1[j] = 0;
      // invertovanje
      for( j = 0; j < pSlika->m_nSirina; j++)
            cPolinom1[j] = cPolinom1[m_nSirina-1-j]?0: 1;
      Mult(pSlika->m_nSirina, cPolinom1,
                     cPolinom2, cBrojUkupnih);
     // NAPOMENA: cBrojUkupnih ne govori koliko je ukupno
     //tacaka, vec na kojim mjestima
     // Ima viska tacaka (van poklapanja)
      for(j=0; j < pSlika->m_nSirina - m_nSirina + 1; j++)
      {
            nG[j]+=cBrojZajednickih[pSlika->m_nSirina+j-1];
            // m_nSirina - cBrojUkupnih   je STVARNI broj
            // ukuponih tacaka (u toj vrsti)
            nD[j] += m_nSirina -
                  cBrojUkupnih[pSlika->m_nSirina + j - 1];
      }
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      // Racunanje mjera slicnosti
      for(j = 0; j < pSlika->m_nSirina - m_nSirina + 1; j++)
            if(nG[j] / nD[j] > nPragTacnosti)
            {
            // U pitanju su izdvojene slike pa je nD[j] != 0!
                  nPojavljivanja[nBr] = j;
                  nBr++;
            }
       return nBr;
}


// Postavlja dvije tacke podjele u nekoj rijeci...
// Ako vec postoji, nece je postaviti
// Vraca broj novih rijeci i (bocno) njihove pozicije
int CSlika::PostaviTackeSegmentacije(int nOd, int nDuzina, int
nPocetak[], int nKraj[])
{
      // U listi su vec uredjeni po velicini
      int nDruga = nOd + nDuzina - 1;
      int nTacka = nOd;
      int nPrethodna = 0;  // Prethodna tacka podjele
      int nBrojNovihRijeci = 0;
      POSITION pos = m_lstTackeSegmentacije.GetHeadPosition();
      while(pos != NULL)
      {
        POSITION posTmp = pos;
        int lstTackaPodjele =
              m_lstTackeSegmentacije.GetNext(pos);
           if(nTacka <= nTackaPodjele)
           {// Umece se, ako vec ne postoji
                if(nTackaPodjele - nTacka > SIRINA_SLOVA
                      && nTacka - nPrethodna > SIRINA_SLOVA )
                {
                     m_lstTackeSegmentacije.InsertBefore(
                                              posTmp, nTacka);
                     nPocetak[nBrojNovihRijeci] = nPrethodna;
                     nKraj[nBrojNovihRijeci] = nTacka;
                     nBrojNovihRijeci++;
                     if(nTacka + nDuzina - 1 + SIRINA_SLOVA <
                             nTackaPodjele && nTacka != nDruga)
                     {
                        nPocetak[nBrojNovihRijeci] = nTacka;
                        nKraj[nBrojNovihRijeci] =
                                       nTacka + nDuzina - 1;
                        nBrojNovihRijeci++;
                        return nBrojNovihRijeci;
                     }
```

Slika 5.2 (nastavak): Funkcije klase *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
                    if(nTacka + nDuzina - 1 <= nTackaPodjele
                                          && nTacka != nDruga)
                          return nBrojNovihRijeci;
                     if(nTacka + nDuzina - 1 >
                           nTackaPodjele + SIRINA_SLOVA
                                    && nTacka != nDruga)
                     {
                       nPocetak[nBrojNovihRijeci] = nTacka;
                       nKraj[nBrojNovihRijeci]= nTackaPodjele;
                       nBrojNovihRijeci++;
                       // Napravljene su dvije nove rijeci,
                       // samo sa prvom tackom
                     }
                }
                if(nTacka == nDruga)
                      return nBrojNovihRijeci;
                 // Postavljena je i krajnja tacka...
                nTacka = nDruga;  // Postavljamo i drugu..
                nDuzina = 0;
          }
          nPrethodna = nTackaPodjele;
     }
     return nBrojNovihRijeci;
 }

bool CSlika::EkvivalentnaSlika(CSlika* pSlika, int nTacnost)
{
     if(m_nSirina > pSlika->m_nSirina+BALANS ||
                   pSlika->m_nSirina > m_nSirina + BALANS)
          return false; // Razlicitih sirina => nisu jednake

     if(m_nSirina <= pSlika->m_nSirina )
          return (Ekvivalentne(pSlika, 0) > nTacnost);
     if(pSlika->m_nSirina < m_nSirina)
          return (pSlika->Ekvivalentne(this, 0) > nTacnost);
     return false;
}
```

Slika 5.2 (nastavak): Funkcije klase *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Ekvivalencija slike sa dijelom druge slike, pocev od zadate
// pozicije vraca mjeru slicnosti (0 - 100).
int CSlika::Ekvivalentne(CSlika* pSlika, int nOd)
{
      int nBajtOd = nOd / 8;
      int nBajtDo = (nOd + m_nSirina - 1) / 8;
      int nShiftL = nOd % 8;
      int nShiftD = 8 - nShiftL;

      int nBrZajednickih = 0;
      int nBrUkupnih = 0;

      for(int i=0; i < pSlika->m_nVisina && i < m_nVisina; i++)
      {
            char cNovi;
            char cZajednicke;
            char cUkupne;
            for(int j = nBajtOd; j < nBajtDo; j++)
            {
                  cNovi = (pSlika->m_nTacka8[i][j] << nShiftL)
                        | (pSlika->m_nTacka8[i][j+1] >> nShiftD);

                  cZajednicke = m_nTacka8[i][j-nBajtOd] & cNovi;
                  cUkupne = m_nTacka8[i][j-nBajtOd] | cNovi;

                  for(unsigned char x = cZajednicke; x;
                                          ++nBrZajednickih)
                        x &= x-1; // brzo sabiranje bitova u bajtu
                  for(x = cUkupne; x; ++nBrUkupnih)
                        x &= x-1; // brzo sabiranje bitova u bajtu
            }
            //Poslednji ima samo nule iza...
            cNovi = (pSlika->m_nTacka8[i][nBajtDo] << nShiftL);
            cZajednicke = m_nTacka8[i][nBajtDo-nBajtOd] & cNovi;
            cUkupne = m_nTacka8[i][nBajtDo-nBajtOd] | cNovi;

            for(unsigned char x=cZajednicke; x; ++nBrZajednickih)
                  x &= x-1; // brzo sabiranje bitova u bajtu
            for(x = cUkupne; x; ++nBrUkupnih)
                  x &= x-1; // brzo sabiranje bitova u bajtu

            }
      if(nBrUkupnih==0)
            return 100; // prazno!!!

      return (int) 100 * nBrZajednickih / nBrUkupnih;
}
```

Слика 5.2 (наставак): Функције класе *CSlika*

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

У првој верзији програма тачка се памтила једним бајтом. Међутим, осим што је програм захтијевао много више меморије, радио је и много спорије.

Ако би се, и код овако представљених података, приступ тачки вршио путем функција *Get* и *Set* то би процес препознавања такође био спор. Такође, и да постоји тип *bool* програм би радио споро. Али, у неколико критичних случајева писане су специјалне рутине за бржи рад. На примјер, на више мјеста се користи рутина за брзо сабирање битова у бајту, упоређивање двије слике се не врши тачка по тачка већ бајт по бајт што смањује приступ тачки 8 пута итд.

У имплементацији преклапања слика преко множења полинома користио се рекурзивни алгоритам сложености $O(n^{\log 3})$. Глобална функција *Mult(int n, char p[], char q[], char r[])* дата је на слици 5.3. Како рекурзивна алокација меморије узима много времена, то је направљена рутина са коришћењем глобалне меморије.

```
// Rekurzivna rutina za mnozenje polinoma slozenosti O(n^log3)
// Koristi globalnu alokaciju memorije

#define LOW     0
#define MIDDLE  1
#define HI      2
#define STEPEN_POL    10000
#define LOG2          16

int ra[LOG2 * 3][STEPEN_POL];  // log2 nivoa za sve (3 puta)
int p[LOG2 * 3][STEPEN_POL], q[LOG2 * 3][STEPEN_POL];

void Mult(int n, int p1[], int q1[], int r[])
{
  for(int i = 0; i < n; i++)
  {
    p[0][i] = p1[i];
    q[0][i] = q1[i];
  }
  MultPomocni(n, 0, 0);
  for(i = 0; i< n; i++)
    r[i] = ra[0][i];
}
```

Slika 5.3: Множење полинома

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Pomocno mnozenje sa globalnim ulazom i izlazom
void MultPomocni(int n, int nNivo, int nDio)
{
  if(n == 1)
  {
    ra[3*nNivo+nDio][0] = p[3*nNivo+nDio][0]*q[3*nNivo+nDio][0];
    return;
  }
  if(n != 2 * ((int) n / 2))
  {
    p[ 3 * nNivo + nDio][n] = 0;
    q[ 3 * nNivo + nDio][n] = 0;
    ++n;
  }
  int n2 = n / 2;
  int nNoviNivo = nNivo + 1;
  for(int i = 0; i < n2; i++)
  {
    p[3 * nNoviNivo + LOW][i] = p[3 * nNivo + nDio][i];
    q[3 * nNoviNivo + LOW][i] = q[3 * nNivo + nDio][i];
  }
  for(i = n2; i < n; i++)
  {
    p[3 * nNoviNivo + HI][i-n2] = p[3 * nNivo + nDio][i];
    q[3 * nNoviNivo + HI][i-n2] = q[3 * nNivo + nDio][i];
  }
  for(i = 0; i < n2; i++)
  {
    p[3*nNoviNivo+MIDDLE][i] =
              p[3*nNoviNivo+LOW][i] + p[3*nNoviNivo+HI][i];
    q[3*nNoviNivo+MIDDLE][i] =
              q[3*nNoviNivo+LOW][i] + q[3*nNoviNivo+HI][i];

  }
  MultPomocni(n2, nNoviNivo, LOW);
  MultPomocni(n2, nNoviNivo, MIDDLE);
  MultPomocni(n2, nNoviNivo, HI);
  for(i = 0; i<n-1; i++)
    ra[3 * nNivo + nDio][i] = ra[3 * nNoviNivo + LOW][i];
  ra[3*nNivo+nDio][n-1] = 0;
  for(i = 0; i<n-1; i++)
    ra[3*nNivo+nDio][n+i] = ra[3*nNoviNivo+HI][i];
  for(i = 0; i<n-1; i++)
    ra[3*nNivo+nDio][n2+i] += ra[3*nNoviNivo+MIDDLE][i]-
              ra[3*nNoviNivo+LOW][i]-ra[3*nNoviNivo+HI][i];
}
```

Slika 5.3: Множење полинома (наставак) помоћна функција

5.2 Сегментација

Најтежи дио у процесу препознавања текста је подјела текста на слова. У глави 3.2 је било ријечи једноставној и сложеној сегментацији.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Једноставна сегментација почива на пројекцији, а слочежа на тражењу подријечи. На слици 5.4 дат је кôд за сложену сегментацију и кластеризацију ријечи.

```
// Slozena podjela rijeci na slova - preko klasterizacije
void  CYuOcrView::SlozenaPodijelaNaSlova(CNizSlika& slova)
{
  CYuOcrDoc* pDoc = GetDocument();
  CNizSlika S;
  CNizSlika Q;
  IzvrsiKlasterizaciju(pDoc->poluRijeci);
  for(int i = 0; i < m_nBrojKlastera; i++)
  {
    CNizSlika* klaster = &m_nsKlaster[i];
    POSITION pos = klaster->GetHeadPosition();
    CSlika* prototip = klaster->GetNext(pos);
    S.AddTail(prototip);
  }
  POSITION pos = S.GetHeadPosition();
  while( pos != NULL)
  {
    CSlika* tmp = S.GetNext(pos);
    // model za rijec
    CSlika* p = new CSlika(tmp->m_nSirina, tmp->m_nVisina);
    for(int i = 0; i < tmp->m_nVisina; i++)
    for(int j = 0; j < tmp->m_nBrBajta; j++)
    p->m_nTacka8[i][j] = tmp->m_nTacka8[i][j];
    Q.AddTail(p);
  }

  // podjela prototipa
  pos = Q.GetHeadPosition();
  while( pos != NULL && !Q.IsEmpty() )
  {
    POSITION posTmp = NULL;
    CSlika* p = Q.GetHead(); // Prvi element iz reda prototipova
    POSITION pos1 = S.GetHeadPosition();
    while( pos1 != NULL)
    {
      CSlika* Ci = S.GetNext(pos1);
      int nOdakle = 0;
      int nPoc[5], nKraj[5];
      if(pDoc->m_bPrekoPolinoma)
      {
        int nPojavljivanja[50];
        int nBrojPojavljivanja =
        p->PodslikaPrekoPolinoma(Ci,pDoc->m_nTacnostSeg,
                                    nPojavljivanja);
```

Слика 5.4: Сложена сегментација и кластеризација

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      for(int k = 0; k < nBrojPojavljivanja; k++)
      {
        // Jeste podslika
        int nBrNovih = Ci->PostaviTackeSegmentacije(
              nPojavljivanja[k], p->m_nSirina, nPoc, nKraj);

        for(int i = 0; i < nBrNovih; i++)
        {
          // Pravi se nova rijec
          CSlika* fragment = Ci->NapraviPodrijec(
                                          nPoc[i], nKraj[i]);
          fragment->m_bRijec = true;
          posTmp = Q.AddTail(fragment);
        }
      }
    }
    else
    {
      int nPocetak = p->Podslika(Ci, nOdakle,
                                 pDoc->m_nTacnostSeg);
      while(nPocetak >= 0)
      {
        // Jeste podslika
        int nBrNovih = Ci->PostaviTackeSegmentacije(
                       nPocetak, p->m_nSirina, nPoc, nKraj);
        for(int i = 0; i < nBrNovih; i++)
        {
          // Pravi se nova rijec
          CSlika* fragment=
                     Ci->NapraviPodrijec(nPoc[i],nKraj[i]);
          fragment->m_bRijec = true;
          posTmp = Q.AddTail(fragment);
        }
        // Da li ima jos koja podrijec p u rijeci Ci?
        nOdakle = nOdakle + nPocetak + p->m_nSirina;
        nPocetak=p->Podslika(Ci,nOdakle,pDoc->m_nTacnostSeg);
      }
    }
  }
```

Слика 5.4: Сложена сегментација и кластеризација

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    // Izbacivanje iz reda slike (modela) p
    delete p;
    Q.RemoveHead( );
    if(pos == NULL) pos = posTmp;
  }

  // Napravljena je podjela svakog prototipa
  pos = pDoc->poluRijeci.GetHeadPosition();
  int nBrRijeci = 0;
  int nBrPoluRijeci = 0;
  int nPrethodni = 0;
  m_anBrSlovaURijeci[nBrRijeci] = 0;
  while( pos != NULL)
  {
    CSlika* rijec = pDoc->poluRijeci.GetNext(pos);
    // Uzimamo prototip
    CSlika* prototip =
                m_nsKlaster[rijec->m_nBrKlastera].GetHead();
    TackePodjele* lstPodjela =
                  &(prototip->m_lstTackeSegmentacije);

    // Podjela je ista kao kod prototipa
    POSITION posTackeSegmentacije =
                        lstPodjela->GetHeadPosition();
    // Uvijek postoje bar dvije tacke!
    int nPrva = lstPodjela->GetNext(posTackeSegmentacije);
    int nDruga;
    while(posTackeSegmentacije != NULL)
    {
      // Izvrsava se barem jednom, jer svaka rijec
      // ima bar dvije tacke podjele
      nDruga = lstPodjela->GetNext(posTackeSegmentacije);
      CSlika* slovo = prototip->NapraviPodrijec(nPrva, nDruga);
      slova.AddTail(slovo);
      m_anBrSlovaURijeci[nBrRijeci]++; // Novo slovo u rijeci
      nPrva = nDruga + 1;
    }
    nBrPoluRijeci++;
    if(m_anBrPoluRijeciURijeci[nBrRijeci] <=
                                      nBrPoluRijeci-nPrethodni)
    {
      // Nova rijec
      nPrethodni = nBrPoluRijeci;
      nBrRijeci++;
      m_anBrSlovaURijeci[nBrRijeci] = 0;
    }
  }
```

Слика 5.4: Сложена сегментација и кластеризација (наставак)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
  // Dealociranje memorije
  // Ne treba dealocirati rijeci vec ih samo izbaciti iz reda S
  // Bice dealocirani kad se budu brisale rijeci
  S.RemoveAll( );

  for(i = 0; i < m_nBrojKlastera; i++)
  {
    CNizSlika* klaster = &m_nsKlaster[i];
    while (!klaster->IsEmpty())
    {
      CSlika* s = klaster->GetHead();
      if(s->m_bRijec)
        delete klaster->RemoveHead();
      else
        klaster->RemoveHead(); // Bice obrisana kasnije
    }
  }
    m_nBrojKlastera = 0;
}

// Klasterizacija rijeci
// Uzecemo da je prototip svakog klastera prva slika u klasteru
void CYuOcrView::IzvrsiKlasterizaciju(CNizSlika& rijeci)
{
  CYuOcrDoc* pDoc = GetDocument();
  bool bDodat;
  POSITION pos = rijeci.GetHeadPosition();
  while(pos != NULL)
  {
    bDodat = false;
    CSlika* pRijec = rijeci.GetNext(pos);
    for(int i = 0; i < m_nBrojKlastera; i++)
    {
      CNizSlika* pKlaster = &m_nsKlaster[i];
      // Posto postoji klaster, nesto ima u njemu...
      CSlika* prototip = pKlaster->GetHead();
      if( prototip->EkvivalentnaSlika(pRijec,
                                      pDoc->m_nTacnostSeg) )
      {
        pKlaster->AddTail(pRijec);
        pRijec->m_nBrKlastera = i; // Pripada i-tom klasteru.
        bDodat = true;
      }
    }

    if(!bDodat)
    {
      // Dodaje se novi klaster, sa samo jednom slikom
      m_nsKlaster[m_nBrojKlastera].AddTail(pRijec);
      // Pripada tom klasteru.
      pRijec->m_nBrKlastera = m_nBrojKlastera;
      m_nBrojKlastera++;
    }
  }
}
```

Слика 5.4: Сложена сегментација и кластеризација (наставак)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Умјесто тражења поклапања ријечи, како је већ речено, прво се изврши једноставна сегментација чиме су направљене тзв. полуријечи. Даље се траже поклапања полуријечи ("подполуријечи") и процес се наставља до издвајања слова(или довољно уских ријечи која и не морају бити слова већ нпр. парови слова. Коко се у бази примјера памте и слике парова слова која се додирују или преклапају, то ће и у том случају бити могуће препознавање.

5.3 Препознавање текста

Како је већ речено препознавање текста почива на препознавању изолованог слова. Препознавање изолованог слова врши се упоређивањем тог слова са свим словима из базе примјера. Препознати карактер је оно слово из базе примјера које је најсличније изолованом слову.

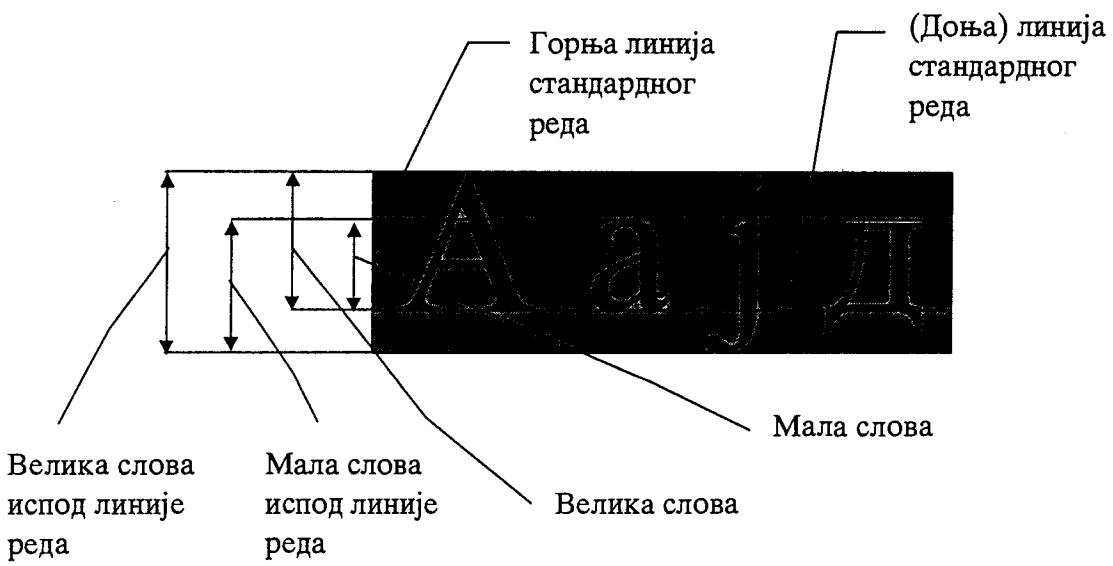
5.3.1 Анализа реда

У српској ћирилици има неколико типова слова која се могу груписати у неколико група. Ако претпоставимо да је слово О репер, тада сва слова могу бити груписана у следеће групе:

- *Велика слова* - слова исте (или бар приближно исте) висине као и слово О. Код ових слова почетак (врх) и крај (дно) по висини су приближно исти као и код слова О.
- *Мала слова* - слова код којих је дно као и код слова О, али је врх нижи.
- *Велика слова испод реда* - слова која су виша од слова О. Врх им је као и слову О, али је дно испод.
- *Мала слова испод реда* - слова којима је врх као и стандардним малим словима, али је ддно испод дна слова О.
- *Специјални знаци* - сви остали знаци (у ствари, преостали су само неки знаци интерпункције)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Претпоставимо, даље, да је висина стандардног реда као и висина слова О. Тада слово, зависно од тога којој групи припада, има различит положај у реду (слика 5.5).



Слика 5.5: Положај слова у реду

У зависности од тога каквог су типа слова у реду ред може имати 4 могуће висине. На основу тога можемо све редове подијелити у 4 класе:

- *RedV* - ред који је висине као и велика слова. У њему се, дакле, налази бар једно велико слово, а не налази ни једно које се повлачи испод линије реда.
- *RedM* - ред који је висине као и мала слова. У њему се не налази ни једно велико слово, као ни једно слово које се повлачи испод линије реда.
- *RedIV* - ред који је висине као неко велико слово које иде испод реда (нпр. Д, ђ и сл.).
- *RedIM* - ред који је висине као неко мало слово које иде испод реда (нпр. р, д и сл.).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Примијетимо да и нека мала слова (ђ, ј, д, и сл.) припадају групи слова *велика слова исūод линије реда*, као и да нека мала (ћ, б, и сл.) припадају групи слова *велика слова.*

Значи, анализирајући висине слова у реду у неким случајевима је могуће одредити тип реда. То је могуће ако има слова разних висина и под претпоставком да су сва слова у реду писана фонтом исте величине.

Тип слова *RedM* није могуће реконструисати на основу висина, јер тада су сва слова у реду исте висине. Отуда се не зна да ли су сва слова велика или сва мала или чак сва иду испод линије реда. Међутим у нормалним текстовима на српском језику практично је немогуће да сва слова у реду буду слова која иду успод линије реда, па у случају да није одређен тип реда можемо претпоставити да је у питању или *tipV* или *tipM.* Према томе, имамо три случаја када је могуће реконструисати тип реда и један када не можемо. Ектракција текста се врши у функцији *OnExtractExtracttext()* класе *CYuOcrView*. Претходно су дефинисане 4 константе које одговарају горњим типовима:

```
#define VELIKA_ISPOD 1
#define VELIKA 2
#define MALA_ISPOD 3
#define NEPOZNATA 4
```

Дио кôда функције *OnExtractExtracttext()* гдје се врши одређивање типа реда дат је на слици 5.6.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
int brReda = 0;
odredjenTipReda = FALSE;
pos = pDoc->slova.GetHeadPosition();
while (pos != NULL)
{
  CSlika* slovo = pDoc->slova.GetNext(pos);
  int vrh, dno, StvarnaVisina;
  int gore;
  int dolje;
  slovo->VrhIDno(&vrh, &dno, &gore, &dolje);
  StvarnaVisina = dno-vrh+1;
  if(!odredjenTipReda)
  if(visinaReda[brReda] - StvarnaVisina >
                          visinaReda[brReda]*0.2)
  //slovo je bitno nize od visine reda-rekonstruisemo tip reda
  if((vrh*2.937<visinaReda[brReda]&&
                    vrh*3.916>visinaReda[brReda])
     ||(dno*1.27<visinaReda[brReda] &&
      dno*1.424>visinaReda[brReda]))
  {
    odredjenTipReda = TRUE;
    klasaReda[brReda] = VELIKA_ISPOD;
  }
  else if(vrh*2.187<visinaReda[brReda]&&
          vrh*2.8>visinaReda[brReda])
  {
     odredjenTipReda = TRUE;
     klasaReda[brReda] = VELIKA;
  }
  else if(dno*1.45<visinaReda[brReda] &&
         dno*1.59>visinaReda[brReda])
  {
     odredjenTipReda = TRUE;
     klasaReda[brReda] = MALA_ISPOD;
  }


  ++kSlova;
  if(kSlova >= brSlovaURijeci[kRijeci]+prethodnaSlova)
  {  // Presli smo u sledecu rijec. Mozda je i novi red
    prethodnaSlova += brSlovaURijeci[kRijeci];
    kRijeci++;
    if(brRijeciURedu[brReda] <= kRijeci - prethodneRijeci)
    {// Novi red!!!!
      prethodneRijeci += brRijeciURedu[brReda];
      if(!odredjenTipReda)
        klasaReda[brReda] = NEPOZNATA;
      brReda++;
      odredjenTipReda = FALSE; // priprema za sledeci red
    }
  }
}
```

Слика 5.6: Одређивање типа реда

Комплетан кôд функције *OnExtractExtracttext()* биће дат касније.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

5.3.2 Базе примјера

Како смо све карактере сврстали у 5 група то ћемо направити 5 база примјера (5 група слова груписаних по висини) . Сваку од ових база ћемо даље рашчланити на одређен број подбаза посматрајући ширину слова. Тако има 7 подбаза базе *VNormal*, 5 подбаза базе *MNormal*, 5 подбаза базе *VIspod*, 5 подбаза базе *MIspod* и 4 подбазе базе *SpecZnaci*.

Базе се чувају у глобалним промјенљивим и слова у свакој бази су већ нормализована ради бржег упоређивања. На слици 5.7. дате су декларације функција за приступ базама и имена слова у свакој од база. Примијетимо да се у процесу сегментације може десити да је неко изоловано слово, у ствари, пар слова. Зато се име сваког слова памти као стринг - за случај када је препознато "слово" пар слова - код слова која се додирују или преклапају.

```
// globalna baza slova
NizSlika VNormal[7];
NizSlika MNormal[5];
NizSlika VIspod[5];
NizSlika MIspod[4];
NizSlika SpecZnaci[4];

NizSlika* NormalVelika(int i);
NizSlika* NormalMala(int i);
NizSlika* IspodVelika(int i);
NizSlika* IspodMala(int i);
NizSlika* SpecijalniZnaci(int i);

static int BrNormalV[] = {1, 17, 22, 18, 26, 62,13};
static int BrNormalM[] = {1, 15, 5, 28, 25};
static int BrIspodV[] = {3, 10, 42, 57, 70};
static int BrIspodM[] = {1, 5, 46, 14};
static int BrSpec[] = {1, 3, 1, 3};

static int SirinaNV[] = {4, 16, 24, 32, 40, 48, 64};
static int SirinaNM[] = {4, 24, 40, 52, 64};
static int SirinaIV[] = {8, 16, 24, 32, 48};
static int SirinaIM[] = {8, 16, 40, 56};
static int SirinaS[] = {4, 8, 4, 32};
static int VisinaS[] = {4, 8, 8, 32};
```

Слика 5.7: Функције за приступ базама примјера и имена слова у базама примјера

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Velika slova standardne (normalne) visine
static CString ImenaNV[]={
      // NormalV1 - super uska slova
      "!",
      // NormalV2 - uska slova
      "/", "?", "1", "2", "3", "4", "5", "6", "7", "8", "9",
      "0", "J", "Z", "b", "#", "$",
      // NormalV3 - nesto uza slova
      "Z", "b", "#", "$", "2", "4", "6", "7", "9", "0", "B",
      "G", "E", "R", "S", "T", "}", "V", "K", "L", "O", "^",
      // NormalV4 -  slova  standardne sirine
      "E", "S", "V", "K", "L", "O","^","A",CString(92),"I", "N",
      "P", "]", "U", "F", "H", "%", "&",
      // NormalV5 -  nesto sira slova
      "%", "@", "Q", "W", "a}", "b}", "bl", "v}", "g}", "}l",
      "}t", "e}", "z}", "i}", "k}", "l}", "n}", "o}", "p}",
      "c}", "t}", "h}", "~}", "Zl", "[", "Rt",
      // NormalV6 -  Siroka slova
      "b}", "}l", "i}", "l}", "n}", "p}", "~}", "[", "bq", "}q",
      "}t", "`}", "q}", "m}", "w}", "{}", "A}", "Al", "Aq",
      "At", "Bl", "Bq", "Vl", "Vq", "Gl", "Gq", "E}",  "El",
      "Eq", "Et", "@l", "Zq", "I}", "Il",   "It", "Kl", "Kq",
      "Ml", "O}", "Ol", "Oq", "Ot", "Pl", "Rq", "Sl", "Sq",
      "St", "Tl", "U}", "Ul", "Uq", "Ut", "Fl", "Ft", "Hl",
      "Ht", "^l", "^q", "Iq", "Pq", "Hq", "^q",
      // NormalV7 -  Super siroka slova
      "Aq", "Oq", "Uq", "@q", "Iq", "Mq", "Pq", "Fq",
      "Hq", "[}", "[l", "[q", "[t"
};
// Mala slova (preostala)
static CString ImenaNM[]={
      // NormalM1 - super uska slova
      ":",
      // NormalM2 - Nesto uza slova
      "a", "v", "g", "e", "z", "i", "k", "l", "n", "o", "p",
      "s", "t", "h", "~",
      // NormalM3 - super uska slova
      "`", "q", "m", "w", "[",
      // NormalM4 - Nesto sira slova
      "al", "at", "vl", "vt","gl","gt","el", "et", "zl", "zt",
      "il", "it", "kl", "kt", "mt","nl","nt", "ol", "ot", "pl",
      "pt", "sl", "st", "tl", "hl", "ht", "~l", "~t",
      // NormalM1 - super siroka slova
      "aq", "vq", "gq", "eq", "`l","`q","`t","zq", "iq", "kq",
      "ml", "mq", "nq", "wl", "wq", "wt","oq","pq", "sq", "tq",
      "hq", "~q", "{l", "{q", "{t"
};
// Velika slova koja idu ispod linije reda (poput D)
static CString ImenaIV[]={
```

Слика 5.7 (наставак): Функције за приступ базама примјера и имена слова у базама примјера

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// IspodV1 - super uska slova
  "j", "(", ")",
  // IspodV2 - uska slova
  "'j", "(j", "f", "aj", "ej", "vj", "gj", "zj", "tj", "sj",
// IspodV3 - nesto uza slova
  "ej", "sj", "vj", "tj",  "D", "C", "X","(u","(r", "(|", "(}",
  "(l", "(t", "a|", "bj", "g|", "dj", "|j","}j","e|","ij", "kj",
  "q", "mj", "w", "oj", "pj", "rj", "s|", "uj", "fj","cj","~j",
  "xj", "Bj", "Gj", "Ej", "Zj", "Rj", "Tj", "Ju", "Sj",
// IspodV4 - slova priblizno iste visine i sirine
  "a|","e|","s|","Ej","Ju","(v","b|","bu","br","v|","d|", "d}",
  "|u", "|r", "|}", "|l", "|t", "}u", "}r", "}|","`j","z|","i|",
  "k|","l|", "qj", "m|", "n|", "wj", "o|", "p|", "r|","p}","t|",
  "u|", "u}", "h|","c|","c}","~|","x|", "x}", "{j", "Aj", "Bu",
  "Gu", "Dj", "E|", "Zu","Zr","Ij","Kj","Q","Oj","Pj","Ru","Uj",
// IspodV5 - siroka slova
  "m|", "Bu", "E|", "Ru", "|q", "`|", "q|","w|","fu","fr", "f|",
  "f}", "fl", "fq", "ft", "{|", "Au", "Ar","A|","Br","Vu","Vr",
  "Gu", "Gr", "Du", "Dr", "Eu", "Er","E|","@u","@r","Iu", "Ir",
  "I|", "Ku", "Kr", "Lu", "Lr", "Qu","Mj","Mu","Mr", "Nu", "Nr",
  "Wu", "Wr", "Ou", "Or", "O|", "Pu", "Pr","Ru","Su","Sr", "Tu",
  "Tr", "]u", "]r", "Ur", "U|", "Fz", "Fr","Hu","Hr","^u","^r",
  "Xu", "Xr", "[u", "[r"
};
// Mala slova koja idu ispod linije reda
// (poput d ili nekih spojenih slova)
static CString ImenaIM[]={
  // IspodM1 - super uska slova
  ";",
  // IspodM2 - uska slova
  "d", "r", "u", "c", "x",
  // IspodM3 - nesto sira slova
  "au", "ar", "vu", "vr","gu","gr","du", "dr","eu", "er", "zu",
  "zr", "iu", "ir", "ku","kr","lu","lr", "nu", "nr", "ou", "or",
  "pu", "pr", "ru", "rl", "rt", "su","sr","tu","tr", "ur", "ul",
  "ut", "hu", "hr", "cu", "cr", "cl","ct","~u","~r", "xu","xr",
  "xl", "xt",
  // IspodM4 - siroka slova
  "`u", "`r", "qu", "qr", "mu", "mr", "wu", "wr", "rq",
  "uq", "cq", "xq", "{u", "{r"
};
// Specijal;ni znaci (malih dimenzija)
static CString ImenaS[]={
      // SpecD - znaci pri dnu reda
      ".",
      // Spec1 - znaci pri vrhu reda
      CString(39),CString(34), "*",
      // Spec1 - znaci ispod linije reda
      CString(44),
      // Spec1 - znaci po sredini reda
      "=", "+", "-"
};
```

Слика 5.7 (наставак): Функције за приступ базама примјера и имена слова у базама примјера

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Базе се приликом стартовања програма учитавају са диска помоћу функције *FormirajBazuPrimjera()*. Кôд функције дат је на слици 5.8.

```
BOOL FormirajBazuPrimjera()
{
  char *pFileNameNV[7];
  pFileNameNV[0] = "NormalV1.dat";
  pFileNameNV[1] = "NormalV2.dat";
  pFileNameNV[2] = "NormalV3.dat";
  pFileNameNV[3] = "NormalV4.dat";
  pFileNameNV[4] = "NormalV5.dat";
  pFileNameNV[5] = "NormalV6.dat";
  pFileNameNV[6] = "NormalV7.dat";

  char *pFileNameNM[5];
  pFileNameNM[0] = "NormalM1.dat";
  pFileNameNM[1] = "NormalM2.dat";
  pFileNameNM[2] = "NormalM3.dat";
  pFileNameNM[3] = "NormalM4.dat";
  pFileNameNM[4] = "NormalM5.dat";

  char *pFileNameIV[5];
  pFileNameIV[0] = "IspodV1.dat";
  pFileNameIV[1] = "IspodV2.dat";
  pFileNameIV[2] = "IspodV3.dat";
  pFileNameIV[3] = "IspodV4.dat";
  pFileNameIV[4] = "IspodV5.dat";

  char *pFileNameIM[4];
  pFileNameIM[0] = "IspodM1.dat";
  pFileNameIM[1] = "IspodM2.dat";
  pFileNameIM[2] = "IspodM3.dat";
  pFileNameIM[3] = "IspodM4.dat";

  char *pFileNameS[4];
  pFileNameS[0] = "SpecD.dat";
  pFileNameS[1] = "SpecG.dat";
  pFileNameS[2] = "SpecI.dat";
  pFileNameS[3] = "SpecS.dat";
```

Слика 5.8: Учитавање база примјера са диска

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
TRY
{
  for(int i=0; i<7; i++)
  {
     CFile baza(pFileNameNV[i], CFile::modeRead);
     for(int k=0; k<BrNormalV[i]; k++)
     {
       CSlika* slovo = new CSlika(SirinaNV[i],32);
       int brBajta = (SirinaNV[i]-1)/8 +1;
       // Uvijek je zauzet makar 1 bajt
       char* buff = new char[32*brBajta];
       baza.Read(buff, 32*brBajta);
       int poz = 0;
       for(int ii=0; ii<32;ii++)
       {
         for(int j=0; j<brBajta; j++)
           slovo->tacka8[ii][j] = buff[poz++];
         slovo->tacka8[ii][brBajta] = 0x00; // poslednji bajt
       }
       VNormal[i].AddTail(slovo);
       delete [32*brBajta]buff;
     }
  }

  // Baza slova NormalM. mala slova
  for(i=0; i<5; i++)
  {
     CFile baza(pFileNameNM[i], CFile::modeRead);
     for(int k=0; k<BrNormalM[i]; k++)
     {
       CSlika* slovo = new CSlika(SirinaNM[i],32);
       int brBajta = (SirinaNM[i]-1)/8 +1;
       // Uvijek je zauzet makar 1 bajt
       char* buff = new char[32*brBajta];
       baza.Read(buff, 32*brBajta);
       int poz = 0;
       for(int ii=0; ii<32;ii++)
       {
         for(int j=0; j<brBajta; j++)
           slovo->tacka8[ii][j] = buff[poz++];
         slovo->tacka8[ii][brBajta] = 0x00;
       }
       MNormal[i].AddTail(slovo);
       delete [32*brBajta]buff;
     }
  }
```

Слика 5.8 (наставак): Учитавање база примјера са диска

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Baza slova NormalM. mala slova
for(i=0; i<5; i++)
{

// Baza slova IspodV. Neka velika slova koja idu
//ispod linije reda
for(i=0; i<5; i++)
{
  CFile baza(pFileNameIV[i], CFile::modeRead);
  for(int k=0; k<BrIspodV[i]; k++)
  {
    CSlika* slovo = new CSlika(SirinaIV[i],32);
    int brBajta = (SirinaIV[i]-1)/8 +1;
    // Uvijek je zauzet makar 1 bajt
    char* buff = new char[32*brBajta];
    baza.Read(buff, 32*brBajta);
    int poz = 0;
    for(int ii=0; ii<32;ii++)
    {
      for(int j=0; j<brBajta; j++)
        slovo->tacka8[ii][j] = buff[poz++];
      slovo->tacka8[ii][brBajta] = 0x00;
    }
    VIspod[i].AddTail(slovo);
    delete [32*brBajta]buff;
  }
}
//Baza slova IspodM.
// Neka mala slova koja idu ispod linije reda
for(i=0; i<4; i++)
{
   CFile baza(pFileNameIM[i], CFile::modeRead);
   for(int k=0; k<BrIspodM[i]; k++)
   {
     CSlika* slovo = new CSlika(SirinaIM[i],32);
     int brBajta = (SirinaIM[i]-1)/8 +1;
     // Uvijek je zauzet makar 1 bajt
     char* buff = new char[32*brBajta];
     baza.Read(buff, 32*brBajta);
    int poz = 0;
    for(int ii=0; ii<32;ii++)
    {
      for(int j=0; j<brBajta; j++)
        slovo->tacka8[ii][j] = buff[poz++];
      slovo->tacka8[ii][brBajta] = 0x00;
    }
    MIspod[i].AddTail(slovo);
    delete [32*brBajta]buff;
   }
}
```

Слика 5.8 (наставак): Учитавање база примјера са диска

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    // Baza slova Spec. Specijalni znaci ax8  a=4 ili a=8
    for(i=0; i<4; i++)
    {
      CFile baza(pFileNameS[i], CFile::modeRead);
      for(int k=0; k<BrSpec[i]; k++)
      {
        CSlika* slovo = new CSlika(SirinaS[i],VisinaS[i]);
        int brBajta = (SirinaS[i]-1)/8 +1;
        // Uvijek je zauzet makar 1 bajt
        char* buff = new char[brBajta*VisinaS[i]];
        baza.Read(buff, brBajta*VisinaS[i]);
        int poz = 0;

    // Baza slova Spec. Specijalni znaci ax8  a=4 ili a=8
    for(i=0; i<4; i++)
        for(int ii=0; ii<VisinaS[i];ii++)
        {
          for(int j=0; j<brBajta; j++)
          slovo->tacka8[ii][j] = buff[poz++];
          slovo->tacka8[ii][brBajta] = 0x00;
        }
        SpecZnaci[i].AddTail(slovo);
        delete [brBajta*VisinaS[i]]buff;
      }
    }
    return TRUE;
  }
  CATCH( CFileException, e )
  {
    #ifdef _DEBUG
      afxDump<<"File could not be opened"<<e->m_cause << "\n";
    #endif
    return FALSE;
  }
  END_CATCH
}
```

Слика 5.8 (наставак): Учитавање база примјера са диска

Базе примјера се учитавају приликом стартовања програма и све вријеме се чувају у глобалним промјењивим - листама поинтера на слике (елементе типа *CSlika*). Приликом завршавања рада меморија се деалоцира позивом функције *DealocirajBaze()*. Кôд фунције је дат на слици 5.9.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
void DealocirajBaze()
{
  for(int i=0; i<7; i++)
  {
    POSITION pos = VNormal[i].GetHeadPosition();
    while (pos != NULL)
      delete VNormal[i].GetNext(pos);
  }
  for(i=0; i<5; i++)
  {
    POSITION pos = MNormal[i].GetHeadPosition();
    while (pos != NULL)
      delete MNormal[i].GetNext(pos);
  }
  for(i=0; i<5; i++)
  {
   POSITION pos = VIspod[i].GetHeadPosition();
   while (pos != NULL)
     delete VIspod[i].GetNext(pos);
  }
  for(i=0; i<4; i++)
  {
    POSITION pos = MIspod[i].GetHeadPosition();
    while (pos != NULL)
      delete MIspod[i].GetNext(pos);
  }
  for(i=0; i<4; i++)
  {
    POSITION pos = SpecZnaci[i].GetHeadPosition();
    while (pos != NULL)
      delete SpecZnaci[i].GetNext(pos);
  }
}
```

Слика 5.9: Деалоцирање меморије заузете базама примјера

5.3.3 Препознавање изолованог слова

Препознавање слова врши се упоређивањем изолованог слова са примјрима из базе примјера. Претходно се одреди тип реда и зависно од висине и ширине изолованог слова одређује се којој бази (и подбази!) слово припада. Функција *nadjiBazu()* одређује којој бази слово припада. Функција је дата на слици 5.10.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
int CYuOcrView::nadjiBazu(CSlika* slovo, int tipReda, int
visinaReda)
{
  if(tipReda == NEPOZNATA)
    return -1;
  int pocetak, kraj, gs, ds;
  slovo->VrhIDno(&pocetak, &kraj, &gs, &ds);
  int sirina = slovo->sirina;
  int visina = kraj-pocetak+1;

  switch(tipReda)
  {
  case VELIKA_ISPOD:
    if(visinaReda - visina < 0.14*visinaReda)
    {// slovo je priblizno iste visine kao i citav red
     // neka od baza IspodV
      if(sirina*3 < visinaReda)
        return 13;          // baza IspodV1;
      if(sirina*1.533 < visinaReda)
        return 14;          // baza IspodV2;
      if(sirina*1.15 < visinaReda)
        return 15;          // baza IspodV3;
      if(sirina*0.92 < visinaReda)
        return 16;          // baza IspodV4;
      return 17;            // baza IspodV5;
    }
    if(pocetak*15<visinaReda && kraj*1.35 > visinaReda)
    {// Neko veliko slovo
      if(sirina*4.6 < visinaReda)
        return 1;           // baza NormalV1;
      if(sirina*2.3 < visinaReda)
        return 2;           // baza NormalV2;
      if(sirina*1.53 < visinaReda)
        return 3;           // baza NormalV3;
      if(sirina*1.21 < visinaReda)
        return 4;           // baza NormalV4;
      if(sirina*0.92 < visinaReda)
        return 5;           // baza NormalV5;
      if(sirina*0.657 < visinaReda)
        return 6;           // baza NormalV6;
      return 7;             // baza NormalV7;
    }
    if(visina > visinaReda * 0.45 && kraj*1.15 >= visinaReda)
    {// Neko malo slovo koje se povlaci ispod linije reda
      if(sirina*4.6 < visinaReda)
        return 18;          // baza IspodM1;
      if(sirina*1.8 < visinaReda)
        return 19;          // baza IspodM2;
      if(sirina*0.92 < visinaReda)
        return 20;          // baza IspodM3;
      return 21;            // baza IspodM4;
    }
```

Слика 5.10: Одређивање којој бази слово припада

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    if(visina > visinaReda * 0.45 && kraj*1.5 > visinaReda)
    {// Neko malo slovo (normalno)
      if(sirina*5.75 < visinaReda)
        return 8;          // baza NormalM1;
      if(sirina*1.84 < visinaReda)
        return 9;          // baza NormalM2;
      if(sirina*1.314 < visinaReda)
        return 10;         // baza NormalM3;
      if(sirina*0.901 < visinaReda)
        return 11;         // baza NormalM4;
      return 12;           // baza NormalM5;
    }
    //Inace, specijalni znaci
    if(pocetak<visinaReda/15) // gornji znaci
      return 23;
    if(pocetak*1.53>visinaReda && kraj*1.15>visinaReda)
      // donji znaci ispod linije
      return 24;
    if(pocetak*1.53>visinaReda && kraj*1.15 <= visinaReda)
      // donji znaci
      return 22;
    return 25;      //srednji znaci

  case VELIKA:
    if(pocetak*11<visinaReda && kraj*1.09>visinaReda)
    {          // veliko slovo
      if(sirina*3.6 < visinaReda)
      return 1;
      if(sirina*1.8 < visinaReda)
      return 2;
      if(sirina*1.2 < visinaReda)
      return 3;
      if(sirina*0.947 < visinaReda)
      return 4;
      if(sirina*0.72 < visinaReda)
      return 5;
      if(sirina*0.657 < visinaReda)
      return 6;
      return 7;
    }
    if(visina > visinaReda * 0.5)
    {// malo slovo
      if(sirina*4.5 < visinaReda)
        return 8;
      if(sirina*1.5< visinaReda)
        return 9;
      if(sirina*1.028 < visinaReda)
        return 10;
      if(sirina*0.705 < visinaReda)
        return 11;
      return 12;
    }
```

Слика 5.10 (наставак): Одређивање којој бази слово припада

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    // neki specijalni znak
    if(pocetak*11 < visinaReda)// gornji...
      return 22;
    if(kraj*1.09 > visinaReda)// donji...
      return 23;
    return 25;

  case MALA_ISPOD:
    if(pocetak*11<visinaReda  &&  kraj*1.78>visinaReda)//  malo
slovo
    {
      if(sirina*4.25 < visinaReda)
        return 8;
      if(sirina*1.416 < visinaReda)
        return 9;
      if(sirina*0.971 < visinaReda)
        return 10;
      if(sirina*0.666 < visinaReda)
        return 11;
      return 12;
    }
    if(pocetak*11<visinaReda && kraj*1.21>visinaReda)
    {// malo slovo ispod linije
      if(sirina*3.4 < visinaReda)
        return 18;
      if(sirina*1.36 < visinaReda)
        return 19;
      if(sirina*0.68< visinaReda)
        return 20;
      return 21;
    }
    // .. ili neki specijalan znak (osim gornjih)
    if(kraj*1.21>visinaReda)//ispod linije...
      return 24;
    if(kraj*1.13>visinaReda)//donje...
      return 23;
    return 25;
  }
  return -1;
}
```

Слика 5.10 (наставак): Одређивање којој бази слово припада

Функција враћа -1 ако база није одређена. Тада се препознавање врши тражењем сличних слова у више база. Ако је одређена база упоређивање се врши само са примјерима из те базе. Због тога су направљене двије преклапајуће функције *prepoznajSlovo()* које проналазе три најсличнија кандидата изолованом слову. Такође, у њима се одређује и мјера сличности тих кандидата са изолованим словом.

На слици 5.11 дате су функције *prepoznajSlovo()*. Прва препознаје слово у одређеној бази, а друга претражује више база.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Prepoznaje tri najvjerovatnija kandidata za slovo;
// Vraca mjeru slicnosti najvjerovatnijeg kandidata
void CYuOcrView::prepoznajSlovo(CSlika* slovo, int nBaza,
                int* najboqi,CString* kandidati, int visinaReda)
{
  NizSlika* baza;
  int i;
  int pozicija[3];
  int pocetakBaze = 0;

  switch(nBaza)
  {
    case 1: case 2: case 3: case 4: case 5: case 6: case 7:
      baza = NormalVelika(nBaza);
      prepoznajSlovoUBazi(slovo, baza, SirinaNV[nBaza-1],
                          32, najbolji, pozicija);
      for(i=0; i<nBaza-1; i++)
        pocetakBaze += BrNormalV[i];
      kandidati[0] = ImenaNV[pocetakBaze+pozicija[0]];
      kandidati[1] = ImenaNV[pocetakBaze+pozicija[1]];
      kandidati[2] = ImenaNV[pocetakBaze+pozicija[2]];
      break;
    case 8: case 9: case 10: case 11: case 12:
      baza = NormalMala(nBaza-7);
      prepoznajSlovoUBazi(slovo, baza, SirinaNM[nBaza-8],
                          32, najbolji, pozicija);
      for(i=0; i<nBaza-8; i++)
        pocetakBaze += BrNormalM[i];
      kandidati[0] = ImenaNM[pocetakBaze+pozicija[0]];
      kandidati[1] = ImenaNM[pocetakBaze+pozicija[1]];
      kandidati[2] = ImenaNM[pocetakBaze+pozicija[2]];
      break;
    case 13: case 14: case 15: case 16: case 17:
      baza = IspodVelika(nBaza-12);
      prepoznajSlovoUBazi(slovo, baza, SirinaIV[nBaza-13],
                          32, najbolji, pozicija);
      for(i=0; i<nBaza-13; i++)
        pocetakBaze += BrIspodV[i];
      kandidati[0] = ImenaIV[pocetakBaze+pozicija[0]];
      kandidati[1] = ImenaIV[pocetakBaze+pozicija[1]];
      kandidati[2] = ImenaIV[pocetakBaze+pozicija[2]];
      break;
    case 18: case 19: case 20: case 21:
      baza = IspodMala(nBaza-17);
      prepoznajSlovoUBazi(slovo, baza, SirinaIM[nBaza-18],
                          32, najbolji, pozicija);
      for(i=0; i<nBaza-18; i++)
        pocetakBaze += BrIspodM[i];
      kandidati[0] = ImenaIM[pocetakBaze+pozicija[0]];
      kandidati[1] = ImenaIM[pocetakBaze+pozicija[1]];
      kandidati[2] = ImenaIM[pocetakBaze+pozicija[2]];
      break;
```

Слика 5.11: Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    case 22: case 23: case 24: case 25:
       baza = SpecijalniZnaci(nBaza-21);
       prepoznajSlovoUBazi(slovo, baza, SirinaS[nBaza-22],
      VisinaS[nBaza-22], najbolji, pozicija);
      for(i=0; i<nBaza-23; i++)
        pocetakBaze += BrSpec[i];
      kandidati[0] = ImenaS[pocetakBaze+pozicija[0]];
      kandidati[1] = ImenaS[pocetakBaze+pozicija[1]];
      kandidati[2] = ImenaS[pocetakBaze+pozicija[2]];
      int pocetak, kraj, gs, ds;
      slovo->VrhIDno(&pocetak, &kraj, &gs, &ds);
      int sirina = slovo->sirina;
      int visina = kraj-pocetak+1;
      if(nBaza == 25 && visina < visinaReda * 0.2)
      {// u pitanju je - (tire, minus...)
        kandidati[0] = ImenaS[7];
        najbolji[0] = 100;
      }
      break;
  }
};


// preklapajuca funkcija prepoznajSlovo, za slucaj da se
// nije otkrio tip slova
// analizira se sirina i visina slova!!!
// u ovom slucaju su sva slova iste visine, pa su ili sva velika
// ili sva mala prakticno je nemoguce da sva budu c, dz, i sl.
// ili specijalni znaci
void CYuOcrView::prepoznajSlovo(CSlika* slovo, int* najbolji,
                        CString* kandidati, int visinaReda)
{
  int pozicija[3];

  int pocetak, kraj, gs, ds;
  slovo->VrhIDno(&pocetak, &kraj, &gs, &ds);
  int sirina = slovo->sirina;
  int visina = kraj-pocetak+1;

  if(visina < 0.45 * visinaReda)
  {// specijalni znak
    if(gs == 0 && ds == 0)//srednje
    {
      NizSlika* bazaSS = SpecijalniZnaci(2);
      prepoznajSlovoUBazi(slovo, bazaSS, 32, 32,
                          najbolji, pozicija);
      kandidati[0] = ImenaS[5+pozicija[0]];
      kandidati[1] = ImenaS[5+pozicija[1]];
      kandidati[2] = ImenaS[5+pozicija[2]];
      if(visina < visinaReda * 0.1)
      {// u pitanju je - (tire, minus...)
        kandidati[0] = ImenaS[7];
        najbolji[0] = 100;
      }
      return;
    }
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    if(gs == 0)//donje
    {
      NizSlika* bazaSD = SpecijalniZnaci(1);
      prepoznajSlovoUBazi(slovo,bazaSD,4,4,najbolji,pozicija);
      kandidati[0] = ImenaS[pozicija[0]];
      kandidati[1] = ImenaS[pozicija[0]];
      kandidati[2] = ImenaS[pozicija[0]];    // isti su!
      return;
    }
    // Inace gornje ili srednje(ako su samo mala slova)
    NizSlika* bazaSG = SpecijalniZnaci(2);
    NizSlika* bazaSS = SpecijalniZnaci(4);
    prepoznajSlovoUBazi(slovo,bazaSS,32,32,najbolji, pozicija);
    int n[6];
    CString kand[6];
    n[0] = najbolji[0];
    kand[0] = ImenaS[5 + pozicija[0]];
    n[1] = najbolji[1];
    kand[1] = ImenaS[5 + pozicija[1]];
    n[2] = najbolji[2];
    kand[2] = ImenaS[5 + pozicija[2]];

    prepoznajSlovoUBazi(slovo,bazaSG,8,8,najbolji,pozicija);
    n[3] = najbolji[0];
    kand[3] = ImenaS[1 + pozicija[0]];
    n[4] = najbolji[1];
    kand[4] = ImenaS[1 + pozicija[1]];
    n[5] = najbolji[2];
    kand[5] = ImenaS[1 + pozicija[2]];
    uredi(n,kand,6);
    najbolji[0] = n[0];
    najbolji[1] = n[1];
    najbolji[2] = n[2];
    kandidati[0] = kand[0];
    kandidati[1] = kand[1];
    kandidati[2] = kand[2];
    return;
  }
  if(slovo->sirina * 3.5 < slovo->visina)
  {
    NizSlika* bazaNV = NormalVelika(1);
    NizSlika* bazaNM = NormalMala(1);

    prepoznajSlovoUBazi(slovo,bazaNV,4,32,najbolji,pozicija);
    int n = najbolji[0];
    int p = pozicija[0];
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    prepoznajSlovoUBazi(slovo,bazaNM,4,32,najbolji,pozicija);
    if(n > najbolji[0])
    {
      najbolji[2] = najbolji[1];
      najbolji[1] = najbolji[0];
      najbolji[0] = n;
      kandidati[0] = ImenaNV[p];
      kandidati[1] = ImenaNM[pozicija[0]];
      kandidati[2] = ImenaNM[pozicija[0]];
    }
    else
    {
      najbolji[2] = n;
      najbolji[1] = n;
      kandidati[0] = ImenaNM[pozicija[0]];
      kandidati[1] = ImenaNM[p];
      kandidati[2] = ImenaNM[p];
    }
    return;
  }
  if(slovo->sirina * 1.8  < slovo->visina)
  {
    NizSlika* bazaNV = NormalVelika(2);
    NizSlika* bazaNM = NormalMala(2);

    prepoznajSlovoUBazi(slovo,bazaNV,16,32,najbolji,pozicija);

    int n[6];
    CString kand[6];
    n[0] = najbolji[0];
    kand[0] = ImenaNV[1 + pozicija[0]];
    n[1] = najbolji[1];
    kand[1] = ImenaNV[1 + pozicija[1]];
    n[2] = najbolji[2];
    kand[2] = ImenaNV[1 + pozicija[2]];

    prepoznajSlovoUBazi(slovo,bazaNM,24,32,najbolji,pozicija);

    n[3] = najbolji[0];
    kand[3] = ImenaNM[1 + pozicija[0]];
    n[4] = najbolji[1];
    kand[4] = ImenaNM[1 + pozicija[1]];
    n[5] = najbolji[2];
    kand[5] = ImenaNM[1 + pozicija[2]];
    uredi(n,kand,6);
    najbolji[0] = n[0];
    najbolji[1] = n[1];
    najbolji[2] = n[2];
    kandidati[0] = kand[0];
    kandidati[1] = kand[1];
    kandidati[2] = kand[2];
    return;
  }
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
if(slovo->sirina * 1.2  < slovo->visina)
{
  NizSlika* bazaNV = NormalVelika(3);
  NizSlika* bazaNM = NormalMala(2);
  prepoznajSlovoUBazi(slovo,bazaNV,24,32, najbolji, pozicija);
  int n[6];
  CString kand[6];
  n[0] = najbolji[0];
  kand[0] = ImenaNV[18 + pozicija[0]];
  n[1] = najbolji[1];
  kand[1] = ImenaNV[18 + pozicija[1]];
  n[2] = najbolji[2];
  kand[2] = ImenaNV[18 + pozicija[2]];
  prepoznajSlovoUBazi(slovo,bazaNM,24,32, najbolji, pozicija);
  n[3] = najbolji[0];
  kand[3] = ImenaNM[1 + pozicija[0]];
  n[4] = najbolji[1];
  kand[4] = ImenaNM[1 + pozicija[1]];
  n[5] = najbolji[2];
  kand[5] = ImenaNM[1 + pozicija[2]];
  uredi(n,kand,6);
  najbolji[0] = n[0];
  najbolji[1] = n[1];
  najbolji[2] = n[2];
  kandidati[0] = kand[0];
  kandidati[1] = kand[1];
  kandidati[2] = kand[2];
  return;
}
if(slovo->sirina * 0.8  < slovo->visina)
{
  NizSlika* bazaNV = NormalVelika(4);
  NizSlika* bazaNM = NormalMala(2);
  prepoznajSlovoUBazi(slovo,bazaNV,32,32, najbolji, pozicija);
  int n[6];
  CString kand[6];
  n[0] = najbolji[0];
  kand[0] = ImenaNV[40 + pozicija[0]];
  n[1] = najbolji[1];
  kand[1] = ImenaNV[40 + pozicija[1]];
  n[2] = najbolji[2];
  kand[2] = ImenaNV[40 + pozicija[2]];
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    prepoznajSlovoUBazi(slovo,bazaNM,24,32, najbolji, pozicija);

    n[3] = najbolji[0];
    kand[3] = ImenaNM[1 + pozicija[0]];
    n[4] = najbolji[1];
    kand[4] = ImenaNM[1 + pozicija[1]];
    n[5] = najbolji[2];
    kand[5] = ImenaNM[1 + pozicija[2]];
    uredi(n,kand,6);
    najbolji[0] = n[0];
    najbolji[1] = n[1];
    najbolji[2] = n[2];
    kandidati[0] = kand[0];
    kandidati[1] = kand[1];
    kandidati[2] = kand[2];
    return;
  }

  if(slovo->sirina * 0.72  < slovo->visina)
  {
    NizSlika* bazaNV = NormalVelika(5);
    NizSlika* bazaNM = NormalMala(3);
    //     NizSlika* bazaNM2 = NormalMala(2);
    prepoznajSlovoUBazi(slovo,bazaNV,40,32, najbolji, pozicija);
    int n[6];
    CString kand[6];
    n[0] = najbolji[0];
    kand[0] = ImenaNV[58 + pozicija[0]];
    n[1] = najbolji[1];
    kand[1] = ImenaNV[58 + pozicija[1]];
    n[2] = najbolji[2];
    kand[2] = ImenaNV[58 + pozicija[2]];
    prepoznajSlovoUBazi(slovo,bazaNM,40,32, najbolji, pozicija);
    n[3] = najbolji[0];
    kand[3] = ImenaNM[16 + pozicija[0]];
    n[4] = najbolji[1];
    kand[4] = ImenaNM[16 + pozicija[1]];
    n[5] = najbolji[2];
    kand[5] = ImenaNM[16 + pozicija[2]];
    uredi(n,kand,6);
    najbolji[0] = n[0];
    najbolji[1] = n[1];
    najbolji[2] = n[2];
    kandidati[0] = kand[0];
    kandidati[1] = kand[1];
    kandidati[2] = kand[2];
    return;
  }
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
if(slovo->sirina * 0.514  < slovo->visina)
{
  NizSlika* bazaNV = NormalVelika(6);
  NizSlika* bazaNM = NormalMala(3);
  //     NizSlika* bazaNM2 = NormalMala(2);
  prepoznajSlovoUBazi(slovo,bazaNV,48,32, najbolji, pozicija);
  int n[6];
  CString kand[6];
  n[0] = najbolji[0];
  kand[0] = ImenaNV[84 + pozicija[0]];
  n[1] = najbolji[1];
  kand[1] = ImenaNV[84 + pozicija[1]];
  n[2] = najbolji[2];
  kand[2] = ImenaNV[84 + pozicija[2]];

  prepoznajSlovoUBazi(slovo,bazaNM,40,32, najbolji, pozicija);

  n[3] = najbolji[0];
  kand[3] = ImenaNM[16 + pozicija[0]];
  n[4] = najbolji[1];
  kand[4] = ImenaNM[16 + pozicija[1]];
  n[5] = najbolji[2];
  kand[5] = ImenaNM[16 + pozicija[2]];
  uredi(n,kand,6);
  najbolji[0] = n[0];
  najbolji[1] = n[1];
  najbolji[2] = n[2];
  kandidati[0] = kand[0];
  kandidati[1] = kand[1];
  kandidati[2] = kand[2];
  return;
}
if(slovo->sirina * 0.47  < slovo->visina)
{
  NizSlika* bazaNV = NormalVelika(6);
  NizSlika* bazaNM = NormalMala(4);
  //     NizSlika* bazaNM2 = NormalMala(2);
  prepoznajSlovoUBazi(slovo,bazaNV,48,32, najbolji, pozicija);
  int n[6];
  CString kand[6];
  n[0] = najbolji[0];
  kand[0] = ImenaNV[84 + pozicija[0]];
  n[1] = najbolji[1];
  kand[1] = ImenaNV[84 + pozicija[1]];
  n[2] = najbolji[2];
  kand[2] = ImenaNV[84 + pozicija[2]];
  prepoznajSlovoUBazi(slovo,bazaNM,52,32, najbolji, pozicija);
```

Слика 5.11 (наставак): Препознавање слова

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    n[3] = najbolji[0];
    kand[3] = ImenaNM[21 + pozicija[0]];
    n[4] = najbolji[1];
    kand[4] = ImenaNM[21 + pozicija[1]];
    n[5] = najbolji[2];
    kand[5] = ImenaNM[21 + pozicija[2]];
    uredi(n,kand,6);
    najbolji[0] = n[0];
    najbolji[1] = n[1];
    najbolji[2] = n[2];
    kandidati[0] = kand[0];
    kandidati[1] = kand[1];
    kandidati[2] = kand[2];
  }
  // Inace su najsira slova, bilo mala bilo velika
  NizSlika* bazaNV = NormalVelika(7);
  NizSlika* bazaNM = NormalMala(5);

  prepoznajSlovoUBazi(slovo,bazaNV,64,32, najbolji, pozicija);
  int n[6];
  CString kand[6];
  n[0] = najbolji[0];
  kand[0] = ImenaNV[146 + pozicija[0]];
  n[1] = najbolji[1];
  kand[1] = ImenaNV[146 + pozicija[1]];
  n[2] = najbolji[2];
  kand[2] = ImenaNV[146 + pozicija[2]];

  prepoznajSlovoUBazi(slovo,bazaNM,64, 32, najbolji, pozicija);

  n[3] = najbolji[0];
  kand[3] = ImenaNM[49 + pozicija[0]];
  n[4] = najbolji[1];
  kand[4] = ImenaNM[49 + pozicija[1]];
  n[5] = najbolji[2];
  kand[5] = ImenaNM[49 + pozicija[2]];
  uredi(n,kand,6);
  najbolji[0] = n[0];
  najbolji[1] = n[1];
  najbolji[2] = n[2];
  kandidati[0] = kand[0];
  kandidati[1] = kand[1];
  kandidati[2] = kand[2];
}
```

Слика 5.11(наставак): Препознавање слова

Обије функције са слике 5.11 позивају функцију *prepoznajSlovoUBazi()* гдје се, у ствари, и врши препознавање. Кôд функције дат је на слици 5.12. Слово се прво нормализује на димензије слôва у бази, па тек онда тражи поклапање са њима.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
void CYuOcrView::prepoznajSlovoUBazi(CSlika* slovo, NizSlika*
baza,
    int skaliranjeS, int skaliranjeV, int* najbolji, int*
pozicija)
{
  CYuOcrDoc* pDoc = GetDocument();

  CSlika* NormalizovanoSlovo =
  slovo->Normalizovano(skaliranjeS,skaliranjeV);
  int i = 0; // pozicija aktuelnog primjera iz baze
  najbolji[0]=0; najbolji[1]=0; najbolji[2]=0;
  pozicija[0]=-1; pozicija[1]=-1; pozicija[2]=-1;

  POSITION p = baza->GetHeadPosition();
  while(p != NULL)
  {
    int kandidat;
    CSlika* baznoSlovo = baza->GetNext(p);
    if(!pDoc->slozenoPoklapanje)
      kandidat = NormalizovanoSlovo->Ekviv(baznoSlovo);
    else
      kandidat = NormalizovanoSlovo->SlozenaEkviv(baznoSlovo);
    if(kandidat > najbolji[0])
    {
      pozicija[2] = pozicija[1];
      najbolji[2] = najbolji[1];

      pozicija[1] = pozicija[0];
      najbolji[1] = najbolji[0];

      pozicija[0] = i;
      najbolji[0] = kandidat;
    }
    else if(kandidat > najbolji[1])
    {
      pozicija[2] = pozicija[1];
      najbolji[2] = najbolji[1];

      pozicija[1] = i;
      najbolji[1] = kandidat;
    }
    else if(kandidat > najbolji[2])
    {
      pozicija[2] = i;
      najbolji[2] = kandidat;
    }
    i++;
  }
```

Слика 5.12: препознавање слова у бази

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    delete NormalizovanoSlovo;
    // ako je baza kratka (npr. NormalV1 ima samo 1 element)
    // drugi i treci kandidat su isti sa prvim
    if(pozicija[1] == -1)
    {
      pozicija[1] = pozicija[0];
      najbolji[1] = najbolji[0];
    }
    if(pozicija[2] == -1)
    {
      pozicija[2] = pozicija[1];
      najbolji[2] = najbolji[1];
    }
}
```

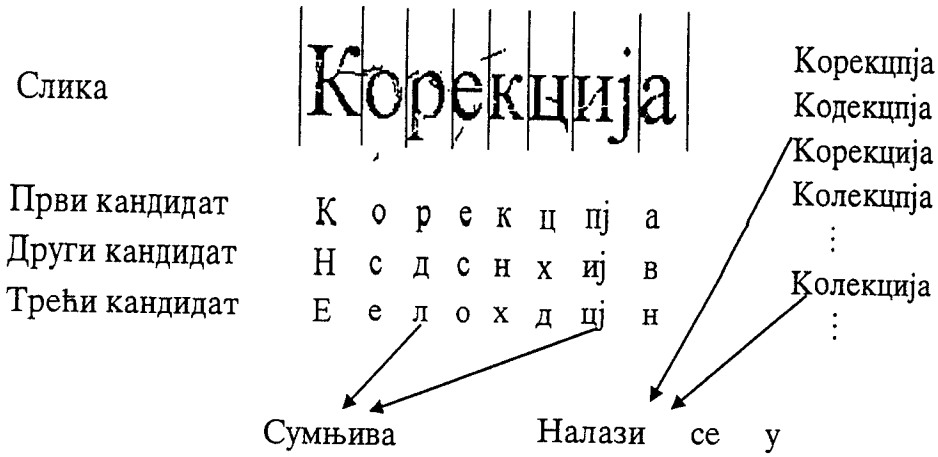
Слика 5.12: препознавање слова у бази

5.4 Корекција грешке

У глави 4 су наведене неке могућности корекције грешке. У програму се врши само лингвистичка корекција грешке. Ријеч код које је неко слово препознато са мањом тачношћу коригује се коришћењем речника. Корекција се врши директном провјером ријечи у речнику. Ради бржег претраживања искоришћено је бинарно претраживање. Само тражење најсличније ријечи углавном се врши коришћењем минамалне дистанце мијењања[9]. У овом раду, за свако изоловано слово дају се три потенцијална кандидата са мјером сличности између кандидата и изолованог слова. Уколико је мјера сличности већа од задате тачности претпоставићемо да је слово исправно препознато. У супротном, то слово означавамо као всумњивог. Ријеч која садржи сумњива слова треба кориговати. Ако поређамо сумњива слова у опадајући низ по мјери сличности и направимо све могуће нове ријечи, за кориговану ћемо прогласити прву од поређаних ријечи која се налази у речнику. Како су сумњива слова поређана у опадајући низ по мјерама сличности, то је коригована ријеч од свих кандидата најсличнија разматраној ријечи (слика 5.13).

[9] minimum edit distance

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 5.13: Корекција грешке помоћу речника. Резултат је ријеч Корекција

На слици 5.14 дат је кôд функција које врше корекцију грешке. Направљена је помоћна функција *kombinacije()* која се користи за тражење кандидата. Такође, како се на крају сваке ријечи може наћи било који знак интерпункције, то се интерпункција на крају ријечи игнорише приликом тражења у речнику.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
//Generise sledecu kobnaciju reda k od n elemenata:1,2,...,n
//Pretpostavlja se da je u nizu komb vec generisana kombinacija
BOOL CYuOcrView::kombinacija(int n, int k, int* komb)
{
  int i;

  i=k;
  while(i > 0 && komb[i] == n-k+i)
    i--;
  if(i==0)
    return NULL;  // nema sledece! komb je poslednja!
  komb[i] = komb[i]+1;
  if(i==k)
    return TRUE;
  for(int j=i+1; j<=k; j++)
    komb[j] = komb[j-1] + 1;
  return TRUE;
}


// trazi rijec u recniku. Ne razmatra se da li su velika ili
mala slova
BOOL CYuOcrView::imaUrecniku(CString rijec)
{
  CString trazenaRijec(rijec);
  trazenaRijec.MakeUpper( );     // prave se velika slova
  int duz = trazenaRijec.GetLength();
  // Ako su poslednja slova znaci interpunkcije,
  // to se ne provjerava u recniku
  int kraj = duz;
  while(--kraj >= 0 && interpunkcija(trazenaRijec[kraj]))
    trazenaRijec = trazenaRijec.Left(kraj);

  NizRijeci* rec = Recnik();

  int poslednji = rec->GetUpperBound( );
  BOOL ima = Find(rec, 0, poslednji, trazenaRijec);

  return ima;
}


// Interpunkcija. Da li je karakter znak interpunkcije
BOOL CYuOcrView::interpunkcija(char znak)
{
  if(znak == (unsigned char)44 || znak == '.' || znak == ';' ||
  znak == ':' ||      znak == (unsigned char)39 || :
  znak == (unsigned char)34 || znak == '!' || znak == '*' ||
  znak == '+' || znak == '-' || znak == '/' || znak == '('||
  znak == ')')
    return TRUE;
  return FALSE;
}
```

Слика 5.14: Функције за корекцију грешке

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Binarno pretrazivanje recnika
BOOL CYuOcrView::Find(NizRijeci* recnik, int poc, int kraj,
                      CString rijec)
{
  if(poc > kraj)
    return FALSE;
  int mid = int((poc + kraj)/2);
  CString predstavnik = (*recnik)[mid];
  if(predstavnik == rijec)
    return TRUE;
  if(rijec < predstavnik)
    return Find(recnik, poc,mid-1, rijec);
  return Find(recnik, mid+1, kraj, rijec);
}

// Koriguje se rijec pomocu recnika.
// rijec je data sa tri pokusaja za svako "sumnjivo" slovo
// pritom, sumnjivo slovo je mozda i par slova!!!
// pozicije sumnjivih slova date su u nizu feler duzine n
// rijec je duzine nn (u stvari, sastoji se od nn stringova!)
BOOL CYuOcrView::koriguj(CString rijeci[30][3], int *feler, int
n, int nn, CString& korigovana)
{
  if(n == 0)
    return FALSE;
  int PocetakSlovaUStringu[30];
  PocetakSlovaUStringu[0] = 0; //prvo slovo pocinje od pozicije
0
  CString rijecZaKorekciju[30][3];
  for(int i=0; i<=nn; i++)
  {
    rijecZaKorekciju[i][0] = rijeci[i][0];
    rijecZaKorekciju[i][1] = rijeci[i][1];
    rijecZaKorekciju[i][2] = rijeci[i][2];
    korigovana += rijeci[i][0];
  }
  if(imaUrecniku(korigovana))
    return TRUE;

  CString novaRijecZaKorekciju[30];
  // Medju sumnjivim se probaju alternative
  // k1 - prva zamjena; k2 - druga zamjena
  int k1,k2;
  int komb1[30], komb2[30];
  for(i=1; i<=n; i++)
    for(k1=i; k1>=0; k1--)
    {
      k2=i-k1;
      for(int a=1; a<=k1; a++)
      komb1[a] = a;
      komb1[0]=0;
      komb1[k1+1]=n+1;
      BOOL p=TRUE;
```

Слика 5.14 (наставак): Функције за корекцију грешке

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      while(p)
      {
        int br=1;
        int slobodni[30];
        for(int a=1; a<=k1+1; a++)
          for(int b=komb1[a-1]+1; b<komb1[a]; b++)
            slobodni[br++]=b;
        for(a=1; a<=k2; a++)
          komb2[a] = a;
        BOOL t = TRUE;
        while(t)
          {
            for(int a=0; a<nn; a++)
              novaRijecZaKorekciju[a] = rijecZaKorekciju[a][0];
            for(int s=1; s<=k1; s++)
              novaRijecZaKorekciju[feler[komb1[s]]] =
              rijecZaKorekciju[feler[komb1[s]]][1];
            for(s=1; s<=k2; s++)
              novaRijecZaKorekciju[feler[slobodni[komb2[s]]]] =
              rijecZaKorekciju[feler[slobodni[komb2[s]]]][2];
            korigovana = "";
            for(a=0; a<nn; a++)
              korigovana += novaRijecZaKorekciju[a];
            if(imaUrecniku(korigovana))
              return TRUE;
            t = kombinacija(n-k1, k2, komb2);
          }
      p = kombinacija(n, k1, komb1);
    }
  }
return FALSE;
}
```

Слика 5.14 (наставак): Функције за корекцију грешке

5.5 Препознавање текста

Процес препознавања текста почиње анализом странице. Страница се дијели на зоне [22] - одваја се текст од слика. Затим се текст дијели на редове, ријечи и слова. У овом раду није разматрана подјела на зоне (издвајање пасуса, наслова, слика....), већ смо се бавили искључиво препознавањем текста.

Позивом функције *OnExtractExtracttext()*почиње процес препознавања у програму. У функцији се обавља подјела на редове, ријечи и слова, ако је речено и корекција грешке итд. Комплетан кôд функције дат је на слици 5.15.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
void CYuOcrView::OnExtractExtracttext()
{
  int brr = 1;
  extract = !extract;

  CYuOcrDoc* pDoc = GetDocument();
  pDoc->BrojSlova = 0;

  if(pDoc->UkljucenTekst)
    return;
  CMainFrame* pFrame = (CMainFrame*) AfxGetMainWnd();
  CStatusBar* pStatusBar =
  (CStatusBar*) pFrame->GetDescendantWindow(AFX_IDW_STATUS_BAR);
  CRect MyRect;
  // Postavlja se na prvi pano
  pStatusBar->GetItemRect(1, &MyRect);
  CProgressCtrl m_Progress;

  pDoc->UkljucenTekst = TRUE;
  pDoc->WndTekst = TRUE;

  //Kreiranje progress kontrole
  m_Progress.Create(WS_VISIBLE|WS_CHILD, MyRect, pStatusBar, 1);

  CSlika* mapa = NapraviSliku(pDoc->GetHDIB());
  if(mapa == NULL)
    return;

  // Racuna se praznina izmedju rijeci,
  // ako nije rucno postavljena
  if(!bPrazno)
    praznina = 6 + 6*pDoc->GetDocSize().cx / 1200;

  mapa->PodijeliNaRedove(pDoc->redovi);
  delete mapa;

  int brRedova = pDoc->redovi.GetCount(); // Koliko ima redova?

  int brRijeciURedu[100 // Koliko ima rijeci u svakom redu?
  int klasaReda[100]; // Kakav je red (po visini)?
  int visinaReda[100]; // Kolika je visina svakog reda?
  int prazninaUredu[100]; // Kolika je praznina izmedju rijeci
  int k = 0;
  int prethodni = 0;

  POSITION pos = pDoc->redovi.GetHeadPosition();
```

Слика 5.15 : Ектракција текста

```
while (pos != NULL)
{
  CSlika* red = pDoc->redovi.GetNext(pos);
  visinaReda[k] = red->visina;
  prazninaUredu[k] = red->visina/5;
  if(bPrazno)
  red->PodijeliNaRijeci(pDoc->rijeci, praznina);
  else
  red->PodijeliNaRijeci(pDoc->rijeci, prazninaUredu[k]);
  brRijeciURedu[k] = pDoc->rijeci.GetCount() - prethodni;
  prethodni += brRijeciURedu[k++];
}
// Koliko ima rijeci ukupno?
int brRijeci = pDoc->rijeci.GetCount();

m_Progress.SetRange(0,brRijeci);
m_Progress.SetStep(1);
m_Progress.StepIt();
// broj slova u rijeci je broj odvojenih simbola.
// moguce da je neki simbol, u stvari, par slova!
int* brSlovaURijeci = new int[brRijeci];
k = 0;
prethodni = 0;
pos = pDoc->rijeci.GetHeadPosition();
if(pDoc->m_nTipSegentacije != 0)
{
  // Slozena segmentacija
  while (pos != NULL)
  {
    // Pravljenje "polurijeci"
    CSlika* rijec = pDoc->rijeci.GetNext(pos);
    rijec->PodijeliNaSlova(pDoc->poluRijeci);
    m_anBrPoluRijeciURijeci[k] = pDoc->poluRijeci.GetCount()
                                          - prethodni;
    prethodni += m_anBrPoluRijeciURijeci[k++];
  }

  SlozenaPodijelaNaSlova(pDoc->slova);
}
else // Jednostavna segmentacija
  while (pos != NULL)
  {
    CSlika* rijec = pDoc->rijeci.GetNext(pos);
    rijec->PodijeliNaSlova(pDoc->slova);
    m_anBrSlovaURijeci[k] =
                pDoc->slova.GetCount() - prethodni;
    prethodni += m_anBrSlovaURijeci[k++];
  }

//Koliko ima slova ukupno?
int brSlova = pDoc->slova.GetCount();

//Odredjivanje tipa reda za svaki red
int kSlova = 0;       // nulto(tj. prvo) slovo
int kRijeci = 0;  // nulta(tj. prva) rijec
int kReda = 0;  // nulti(tj. prvi) red
```

Слика 5.15 (наставак) : Ектракција текста

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
// Koliko je rijeci u prethodnim redovima?
int prethodneRijeci = 0;
// Koliko je slova u prethodnim rijecima?
int prethodnaSlova = 0;

int pozicijaSlova = 0;
BOOL odredjenTipReda = FALSE;

int brReda = 0;

pos = pDoc->slova.GetHeadPosition();
while (pos != NULL)
{
  CSlika* slovo = pDoc->slova.GetNext(pos);
  int vrh, dno, StvarnaVisina;
  int gore;
  int dolje;
  slovo->VrhIDno(&vrh, &dno, &gore, &dolje);
  StvarnaVisina = dno-vrh+1;
  if(!odredjenTipReda)
    if(visinaReda[brReda]-StvarnaVisina >
                                    visinaReda[brReda]*0.2))
      if((vrh*2.937<visinaReda[brReda]&&
          vrh*3.916>visinaReda[brReda]) ||
          (dno*1.27<visinaReda[brReda] &&
          dno*1.424>visinaReda[brReda]))
      {
        odredjenTipReda = TRUE;
        klasaReda[brReda] = VELIKA_ISPOD;
      }
      else if(vrh*2.187<visinaReda[brReda] &&
              vrh*2.8>visinaReda[brReda])
      {
        odredjenTipReda = TRUE;
        klasaReda[brReda] = VELIKA;
      }
      else if(dno*1.45<visinaReda[brReda] &&
              dno*1.59>visinaReda[brReda])
      {
        odredjenTipReda = TRUE;
        klasaReda[brReda] = MALA_ISPOD;
      }

  ++kSlova;
  if(kSlova >= brSlovaURijeci[kRijeci]+prethodnaSlova)
  {  // Presli smo u sledecu rijec. Mozda je i novi red
    prethodnaSlova += brSlovaURijeci[kRijeci];
    kRijeci++;
```

Слика 5.15 (наставак) : Ектракција текста

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      if(brRijeciURedu[brReda] <= kRijeci - prethodneRijeci)
      {// Novi red!!!!
        prethodneRijeci += brRijeciURedu[brReda];
        if(!odredjenTipReda)
           klasaReda[brReda] = NEPOZNATA;
        brReda++;
        odredjenTipReda = FALSE; // priprema za sledeci red
      }
    }
  }

  // Kreiranje novog prozora - za tekst sa istim naslovom + ext
  CString naslov = pDoc->GetTitle();//, strBaseName, strExt;
  int poz;
  naslov = naslov.Mid(naslov.Find('-')+1);
  if((poz = naslov.Find('.')) == -1)
     naslov = naslov+".txt";
  else
     naslov = naslov.Left(poz)+".txt";
  pFrame->tekuciNaslov = naslov;
  pDoc->OnViewOutput();

  kSlova = 0;
  kRijeci = 0;
  kReda = 0;
  prethodneRijeci = 0;
  prethodnaSlova = 0;
  COLORREF boja;
  int pozSlova, pozSumnjivogSlova;
  int sumnjivoSlovo[30];
  CString KandidatRijec[30][3];
  pozSlova = 0;
  pozSumnjivogSlova = 0;

  pos = pDoc->slova.GetHeadPosition();
  while (pos != NULL)
  {
    CSlika* slovo = pDoc->slova.GetNext(pos);

    // Odredjivanje baza
    // Odrediti prvo visinu reda
    int baza=nadjiBazu(slovo,
                  klasaReda[kReda], visinaReda[kReda]);
    CString kojeSlovo[3];
    int najbolji[3];
    if(baza!= -1)
    prepoznajSlovo(slovo, baza, najbolji,
                  kojeSlovo, visinaReda[kReda]);
    else
      prepoznajSlovo(slovo,najbolji,kojeSlovo,
                                visinaReda[kReda]);
```

Слика 5.15 (наставак) : Ектракција текста

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
if(najbolji[0] < pDoc->tacnostChar)
{
  ++pozSumnjivogSlova;
  if(pozSumnjivogSlova == 30)
  {// Prekoracenje duzine rijeci (duza je od 30)
   delete [brRijeci]brSlovaURijeci;
   CString strMsg="Praznina je lose podesena!!!";
   MessageBox(strMsg,"Prekid ektrakcije",
                                    MB_ICONHAND|MB_OK);
   return;
  }
  sumnjivoSlovo[pozSumnjivogSlova] = pozSlova;
}
KandidatRijec[pozSlova][0] = kojeSlovo[0];
KandidatRijec[pozSlova][1] = kojeSlovo[1];
KandidatRijec[pozSlova++][2] = kojeSlovo[2];
++kSlova;
if(kSlova >= brSlovaURijeci[kRijeci]+prethodnaSlova)
{  // Presli smo u sledecu rijec
  if(vocabulary)
  {
    CString ispravljena;
    if(koriguj(KandidatRijec,sumnjivoSlovo,
              pozSumnjivogSlova, pozSlova, ispravljena))
    {
      pFrame->OnOutputText(ispravljena, RGB(0,0,0), 400);
      GetLength( );
      for(int i = 0; i < ispravljena. GetLength( ); i++)
      pDoc->IzlazniString[pDoc->BrojSlova++] =
                                ispravljena[i];
    }
    else
    {   // zadrzava se stara nepopravljena rijec (u boji!)
      int kk=1;
      for(int i=0; i<pozSlova; i++)
      {
        if(sumnjivoSlovo[kk] != i)
          boja = RGB(0, 0, 0);    // Crna
        else
        {
          k++;
          boja = RGB(255, 0, 0); // crvena
        }
        pFrame->OnOutputText(KandidatRijec[i][0],boja, 400);
       for(int ii=0;ii<KandidatRijec[i][0].GetLength();ii++)
        pDoc->IzlazniString[pDoc->BrojSlova++] =
                                    KandidatRijec[i][0][ii];
      }
    }
  }
```

Слика 5.15 (наставак) : Ектракција текста

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      else
      {     // zadrzava se stara nepopravljena rijec (u boji!)
        int kk=1;
        for(int i=0; i<pozSlova; i++)
        {
          if(sumnjivoSlovo[kk++] != i)
            boja = RGB(0, 0, 0); // Crna
          else
            boja = RGB(255, 0, 0); // crvena
          pFrame->OnOutputText(KandidatRijec[i][0], boja, 400);
          for(int ii=0;ii<KandidatRijec[i][0].GetLength();ii++)
            pDoc->IzlazniString[pDoc->BrojSlova++] =
                                        KandidatRijec[i][0][ii];
        }
      }
      ASSERT(kRijeci<brRijeci);
      prethodnaSlova += brSlovaURijeci[kRijeci];
      kRijeci++;
      if(brRijeciURedu[kReda] <= kRijeci - prethodneRijeci)
      { // Novi red!!!!
        ASSERT(kReda<brRedova);
        pFrame->OnOutputText("\r\n", boja, 400);
        pDoc->IzlazniString[pDoc->BrojSlova++] = '\r';
        pDoc->IzlazniString[pDoc->BrojSlova++] = '\n';

        prethodneRijeci += brRijeciURedu[kReda];
        kReda++;
        pozSlova = 0;
        pozSumnjivogSlova = 0;
      }
      else  // Nije novi red, ali je nova rijec
      {
        pFrame->OnOutputText(" ", boja, 400);
        pDoc->IzlazniString[pDoc->BrojSlova++] = ' ';
        pozSlova = 0;
        pozSumnjivogSlova = 0;
      }
      m_Progress.StepIt();
    }
  }

  // Razna dealociranja memorije
  pDoc->DealocirajSlike;
  delete [brRijeci]brSlovaURijeci;
}
```

Слика 5.15 (наставак) : Ектракција текста

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

6. Закључак

Папир је као средство размјене информација и знања широко распростањен. С друге стране, информације у електронској форми су згодније и за претраживање и за мијењање. OCR систем представља мост између штампаног документа и одговарајућег електронског записа.

У раду су изложене неке технике препознавања текста и могућност њихове примјене на ћирилицу и српски језик. Разматране су технике које почивају на подјели текста до нивоа слова. Није обрађена подјела текста на зоне [22] (издвајање наслова, пасуса, слика...), већ само подјела "чистог" текста.

Као резултат неких изложених техника написан је програм YuOCR. Перформансе програма је могуће поправити у више праваца. Могуће је направити сложенију сегментацију, поправити препознавање изолованог слова и имплементирати комплекснији речник (статистички речник парова ријечи).

Такође се може имплементирати препознавање наслова, пасуса и других зона у документу, препознавање слика, формула итд. На крају, могуће је додати и "учење" система.

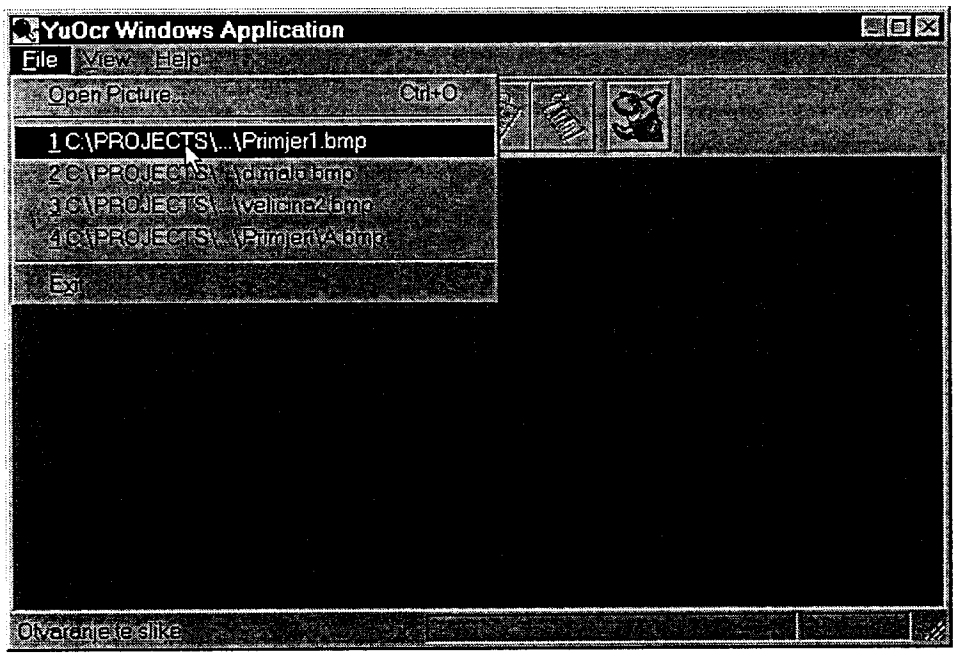
Ипак, мора се имати у виду да иако би свака од наведених смјерница заиста повећала проценат тачно препознатих слова (или ријечи) и препознати текст приближила оригиналу, брзина би се смањила. Стога, морамо тражити компромис између брзине и тачности у процесу препознавања текста.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Додатак А. Упутство за употребу апликације YuOcr

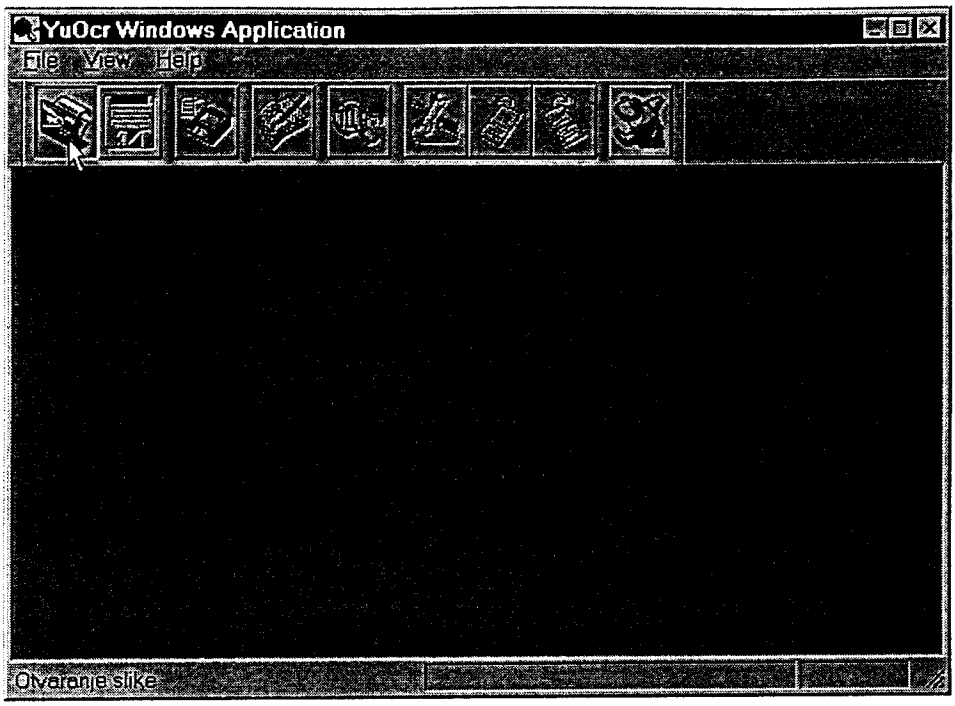
YuOcr је програм за препознавање српске ћирилице. Писан је за оперативни систем WINDOWS 95. Програм врши превођење слике у BMP формату у текст.

Отварање слике врши се из менија *file,open*, или коришћењем одговарајућег дугмета из тулбар-а (слике А.1 и А.2) . Такође је могуће учитати слику и коришћењем комбинације тастера на тастатури *<Ctrl> + O*. На почетку рада то је и једина опција у менију *file*.



Слика А.1: Отварење слике из менија

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика А.2: Отварење слике преко тулбар-а

Екстракција текста (пошто је учитана слика) врши се из менија *Extract, Extract text*, преко одговарајућег дугмета тулбар-а или типке *<Enter>* на тастатури. До подменија *Extract* могуће је доћи и десним кликом миша на прозор слике текста (слика А.3).

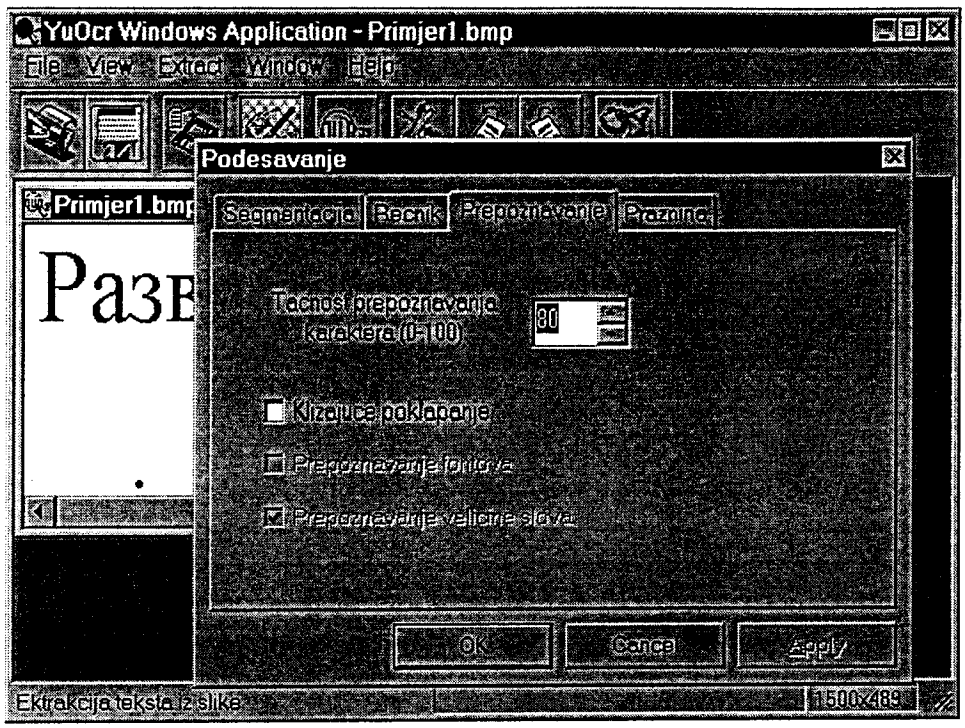
Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Слика А.3: Екстракција текста помоћу падајућег менија

Речник се укључује преко менија *Extract, Recnik*, преко дугмета тулбар-а или преко тастатуре комбинацијом тастера *<Ctrl> + R*.

Разна подешавања се врше из менија *Extract, Podesavanje*, преко одговарајућег дугмета тулбар-а. Овдје је могуће подесити празнину између ријечи и врсту поклапања слова са словом из базе примјера (слика А.4).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика А.4: Подешавања апликације

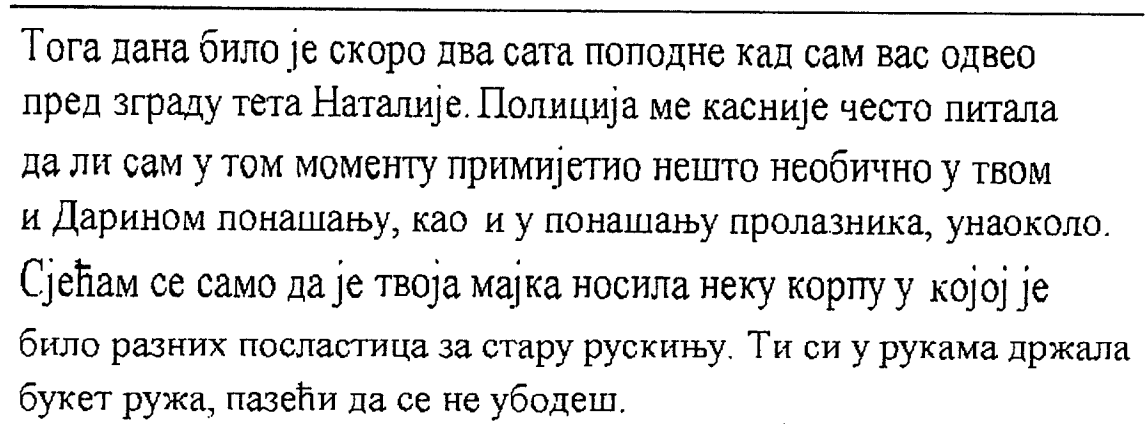
Слику је могуће ротирати улијево или удесно коришћењем опције менија *Extract, Lijeva rotacija*, (односно *Extract, Desna rotacija*), одговарајућих дугмета тулбар-а или, пак, курсорским типкама на тастатури.

Снимање препознатог текста обавља се из менија *File, Save* или одговарајућег дугмета тулбар-а.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Додатак Б. Неке препознате странице

На слици Б.1. приказана је слика једног текста чије се све ријечи налазе у речнику.


Тога дана било је скоро два сата поподне кад сам вас одвео пред зграду тета Наталије. Полиција ме касније често питала да ли сам у том моменту примијетио нешто необично у твом и Дарином понашању, као и у понашању пролазника, унаоколо. Сјећам се само да је твоја мајка носила неку корпу у којој је било разних посластица за стару рускињу. Ти си у рукама држала букет ружа, пазећи да се не убодеш.

Слика Б.1: Слика текста

На слици Б.2 дат је препознати текст без употребе речника и "клизајућег" поклапања, на слици Б.3 са клизајућим поклапањем без употребе речника и на слици Б.4 препознати текст уз употребу "клизајућег" поклапања и речника.

Тога дана било је скоро два сата поподне кад сам вас одвео
пред зграду тета Наталпје. Полицпја ме каснпје често питала
да ли сам у том моменту примпјетио нешто необично у твом
и Дарином понашању, као и у понашању пролазника, унаоколо.
Сјећам се само да је твоја мајка носила неку корпу у којој је
било разних посластица за стару рускињу. Ти си у рукама држала
букет ружа, пазећи да се не убодеш.

Слика Б.2: препознати текст без употребе "клизајућег" поклапања и речника

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Тога дана било је скоро два сата поподне кад сам вас одвео
пред зграду тета Наталпје. Полицпја ме касппје често питала
да ли сам у том моменту примпјетио нешто необично у твом
и Дарином понашању, као и у понашању пролазника, унаоколо.
Сјећам се само да је твоја мајка носила неку корпу у којој је
било разних посластица за стару рускињу. Ти си у рукама држала
букет ружа, пазећи да се не убодеш.

Слика Б.3: препознати текст без употребе речника, а са употребом "клизајућег" поклапања

Тога дана било је скоро два сата поподне кад сам вас одвео
пред зграду тета Наталије. Полиција ме касније често питала
да ли сам у том моменту примијетио нешто необично у твом
и Дарином понашању, као и у понашању пролазника, унаоколо.
Сјећам се само да је твоја мајка носила неку корпу у којој је
било разних посластица за стару рускињу. Ти си у рукама држала
букет ружа, пазећи да се не убодеш.

Слика Б.3: препознати текст са употребом "клизајућег" поклапања и речника

Проценат успјечно препознатих слова у првом случају је 99,07%, у другом 99,07%, а у трећем 100%.

Напоменимо још једном да је ово текст код кога су све ријечи садржане у речнику, па се укључивањем речника битније повећао број препознатих слова. Наравно, у текстовима код којих се налази мали број ријечи из речника препознавање се неће битно поправити.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Литература

[1] Bunke H., A Fast Algorithm for Finding the Nearest Neghbor of a Word in a Dictionary, *tehnical reports IAM-93-025,Institute of Computer Science and Applied Mathematics, Bern*, 1993.

[2] Buse, R., Liu, Z. Q., Caelli, T., "A Structural and Relational Approach to Handwritten Word Recognition", IEEE Trans.Systems, Man, and Cybernetics, Part B, vol. 27, no.5, pp. 847-861, Oct . 1997

[3] Chang, F., Lu, Y., Pavlidis, T., "Feature Analysis Using Line Sweep Thinning Algorithm," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.21, no.2, pp. 145-157, february 1999.

[4] Chang, F., Lu, Y.C. Pavlidis, T.," Line Sweep Thinning Algorithm: A Complete Tehnical Report " TR-IIS-97-011, Institute of Information Science, Academia Sinica, Taipei, Taiwan, 1997.

[5] Cote, M., Cheriet, M., Lecolinet, E., Suen, C. Y.,"Automatic Reading of Cursive Scripts Using Human Knowledge", Technical Report, CENPARMI, Concordia Universitym, Canada, .

[6] Doermann, D., Yao, S., Generating Synthetic Data for Text Analysis Systems. *In Symposium on Document Analysis and InformationRetrieval,* 1995.

[7] Elliman, D.,G., Lancaster, I., K., A review of segmentation and contextual analysis techniques for text recognition. *Pattern Recognition, 23 (3/4), p. 337-346,* 1990

[8] Fang, C., Deciphering Algorithms for Degraded Document Recognition. *Ph.D., State University of New York at Buffalo,* 1995.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

[9] Hew, P. C., "Recognition of Printed Digits Using Strokes, Zernike Moments and Gaussian Models", Technical Report, The University of Western Australia, Oct. 1997.

[10] Hew, P. C., "Single-Sized Template Base OCR", Technical Report, The University of Western Australia, July 1996.

[11] Hew, P. C., "Combining Euclidean and Mahalanobis Distances for Recognition", Technical Report, The University of Western Australia, Feb. 1997.

[12] Hew, P. C.,, "Estimating the Reliability of a Character Recognition System", Technical Report, The University of Western Australia, Nov. 1996.

[13] Hong, T., Degraded Text Recognition Using Visual and Linguistic Context. *Ph.D., State University of New York at Buffalo,* 1995.

[14] Kato, N., Suzuki, M., Omachi, S., Aso, H., Nemoto Y., "A Handwriten Character Recognition System Using Directional Element Feature and Asymmetric Mehalanobis Distance," IEEE Trans. on pattern analzsis and machine intelligence, vol.21, no.3, march 1999.

[15] Khoubyari S., The Application of Word Image Matching in Text Recognition, *A thesis for the degree of Master of Science, University of New York at Buffalo,* 1992.

[16]Liang, S., Ahmadi, M., Shridhar, M., Segmentation of touching characters in printed document recognition. In *Proceding of the Second International Conference on Document Analysis and Recognition ICDAR-94, p. 569-572.* 1993.

[17] Marcelli, A., Likhareva, N., Pavlidis, T., "Structural Indexing for Character Recognition," Computer vision and Image Understanding, vol.66, no.3, pp.330-346, 1997.

[18] Ning Li., An implementation of OCR System Based on Skeleton Matching. *UKC report, Computing Laboratory, University of Kent at Canterbury,* 1991.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

[19] Srihari, S.N., High-Performance Reading Machines, *Proceeding of the IEEE, 80(7), p. 1120-1132.* July 1992.

[20] Stephen, G.A., String Search, *tehnical reports TR-92-gas-01, School of Electronic Engireering Science University College of North Wales*, october 1992.

[21] Sun, F., Omachi, S., Aso, H., "Precise Selection of Candidates for Handwritten Character Recognition Using Feature Regions", Trans. IEICE, vol. E79-D, no. 5, pp. 510-515, May 1996.

[22] Wahl, F.M., Wong, K.Y., Casey, R.G., Block Segmentation and Text Extraction in Mixed Text/Image Documents, *Computer Graphics and Image Processing, 20, p. 375-390*, 1982.

[23] Wakabayashi, T., Deng, Y., Tsuruoka, S., Kimura, F., Miyaki, Y.,"Accuracy Improvment by Nonlinear Normalization and Feature Compression in Handwritten Chinese Character Recognition", Trans. IEICE, vol. J79-D-II, no. 5, pp. 765-774, May 1996.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs