ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

დავით კობახიძე

ქართული ენის ამომცნობი სისტემა

სამაგისტრო პროგრამა - ინფორმაციული ტექნოლოგიები

ნაშრომი შესრულებულია ინფორმაციული ტექნოლოგიების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: პროფ. მაია არჩუაძე

თბილისი

2019

სარჩევი

ანოტაცია

ამ ნაშრომში განხილულია სისტემა, რომელიც ახდენს ქართული ენის ამოცნობას და დამუშავებას სხვადასხვა ინფორმაციული წყაროდან. მაგალითისთვის, როგორიცაა ყველასათვის კარგად ნაცნობი ვიკიპედია, ფეისბუქი და სხვა მრავალი სოციალური და საინფორმაციო ქსელი, მაგრამ ამოცნობისთვის და დამუშავებისთვის არ გამოიყენება სტანდარტული სტემინგის და ლემატიზაციის ალგორითმები, ვინაიდან ქართული ენა და მისი თავისებურებები შეუძლებელს ხდის სტანდარტული ალგორითმების გამოყენებას, ამიტომ საჭირო გახდა სტემინგის ახალი ალგორითმი, რომელიც მოახდენდა ტექსტის დამუშავებას და რომელიც ამ კონკრეტულ შემთხვევაში ეფუძნება მხოლოდ და მხოლოდ სიტყვების საწყისი ფორმების ბაზას.

ასევე უნდა აღინიშნოს რომ ქართული ენისთვის მსგავსი სისტემები არც თუ ისე ბევრია და ამ ნაშრომში ყველა არსებული სისტემის დადებითი მხარე ასე თუ ისე გათვალისწინებულია და გაუმჯობესებულია მათი უარყოფითი მხარეები. სწორედ ამიტომ ამ სისტემამ მოგვცა გარკვეული შედეგები, რომელი შედეგებიც იძლევა პოზიტივის საფუძველს და ამიტომაც შეგვიძლია თამამად ვთქვათ, რომ ეს სისტემა შეიძლება იყოს საწყისი ეტაპი იმისა, რომ დაიწყოს, განვითარდეს და მოხდეს სრულყოფილი სისტემის შექმნა, რომელიც მოახდენს ქართული ენის სრულად ამოცნობას და დამუშავებას.

ANNOTATION

ANNOTATION

In this thesis has been examined a system which implements recognition and processing of the Georgian language from various information sources, for example, such as for all well known Wikipedia, Facebook and other numerous social and information networks, but for recognition and processing are not used the algorithms of standard stemming and lemmatization, as the Georgian language and its peculiarities make use of the standard algorithms impossible, that is why a new stemming algorithm became necessary which would implement the processing of the text and which in this concrete case is based exclusively on the base of the initial forms of conjunctions, pronouns and nouns.

It should also be mentioned that similar systems are not so many for the Georgian language and in this thesis the positive sides of all existing systems, one way or another, are stipulated and their negative sides are improved. Exactly thus and so, this system gave us certain results, these results give us the basis of the positive outcome and that is why we can say with certainty that this system can be an initial stage of commencement, development and creation of a full-fledged system that will implement complete recognition and processing of the Georgian language.

შესავალი

მონაცემთა ბაზის ცნება ამჟამად რამდენადმე გაფარდოვდა, გამომდინარე თანამედროვე ტექნოლოგიების განვითარების ტენდენციებიდან. გაჩნდა ინტერნეტი, რომელიც წარმოადგენს უზარმაზარ ციფრულ ბიბლიოთეკას, რომელშიც ინფორმაციის განთავსება ხდება სხვადასხვა ფორმით (ტექსტური, ვიდეო, აუდიო და ა.შ) რაც ყველაზე მნიშვნელოვანია, ინფორმაცია განთავსებულია არასტრუქტურირებული სახით. ინფორმაციის ყველაზე დიდი წყარო სოციალური ქსელებია, რომლის საშუალებითაც ტერაბაიტების მოცულობის ინფორმაცია ყოველდღიურად განთავსდება ინტერნეტ სივრცეში. ამ პროცესმა წარმოშვა ინფორმაციის სიჭარბის პრობლემა, რომელიც ახლებურად წარმოაჩენს ინფორმაციის ძებნის პროცესს, მეტ მოთხოვნას უყენებს საძიებო სისტემებს და მოითხოვს ძებნის მეთოდების გაუმჯობესებას.

თანამედროვე ინფორმაციული ტექნოლოგიების განვითარებასთან ერთად მას შემდეგ რაც გაჩნდა ინტერნეტი, საჭირო გახდა იმ ინფორმაციის დამუშავება, რომელიც განთავსებულია სოციალურ და საინფორმაციო ქსელებში. ასეთი სახის ინფორმაცია ყოველდღიურად იმატებს და ეს მონაცემები წარმოდგენილია არასტრუქტურირებული სახით, რაც წარმოადგენს ყველაზე დიდ პრობლემას თანამედროვე საინფორმაციო სისტემებში, ვინაიდან ინფორმაციის არასტრუქტურირებული ფორმით განთავსებამ გამოიწვია განსხვავებული მიდგომები მონაცემთა დამუშავების მხრივ. თუ მონაცემთა სტრუქტურირებული ბაზების დამუშავებისათვისაც SQL საკმარისი იყო, დღეს უკვე გვაქვს არასტრუქტურირებული მონაცემები Big Data და მათ დასამუშავებლად NoSQL. სტურქტურირებულ ინფორმაციაში ძებნის შედეგი ზუსტია, არასტრუქტურირებულ ბაზებში უკვე ჩნდება მოთხოვნის შედეგთან შესაბამისობის ხარისხი, რელევანტურობის სიდიდე. ბუნებრივია, მონაცემების სტრუქტურიზაცია გამოიწვევს ძებნის პროცესის გაუმჯობესებას, რაც, პირველ რიგში, აისახება შედეგის რელევანტურობის გაზრდაზე. ძებნის ოპერაციების შესრულებისას მნიშვნელოვანია კიდევ ერთი ფაქტორი: ძირითადად ინფორმაცია განთავსებულია ბუნებრივ სალაპარაკო ენაზე წარმოდგენილი ტექსტების „დოკუმენტების“ სახით და, შესაბამისად, ძებნის სისტემის მუშაობაში გათვალისწინებული უნდა იყოს ბუნებრივი ენის თავისებურებათა მინიმალურად აუცილებელი რაოდენობა.

ამ ნაშრომის მთავარი მიზანიც ესაა, რომ ქართულმა ენამ ფეხი აუწყოს თანამედროვე საინფორმაციო ტექნოლოგიებს და გახდეს ადვილად „ხელმისაწვდომი“ ციფრულ სამყაროში. ნაშრომის მთავარი აზრი მდგომარეობს იმაში რომ შეიქმნას პროდუქტი, რომელიც მოახდენს ქართული ენის ამოცნობას, მის დამუშავებას. სტრუქტურიზაციას და შენახვას შემდგომი გამოყენებისთვის.

ნაშრომი შედგება რამდენიმე თავისგან. რომლებშიც მკითხველი გაეცნობა დღესდღეობით არსებულ მდგომარეობას აღნიშნულ საკითხთან დაკავშირებით, გამოყენებულ თემებსა და ინსტრუმენტებს, რომელთა საშუალებითაც შეიქმნა ეს სისტემა, გაეცნობა კონკრეტულად ამ სისტემის მეთოდებს, საშუალებებს, გზებს რომლებიც გამოვიყენე

პროდუქტის შესაქმნელად, ნახავს სისტემის პრაქტიკულ რეალიზაციას და ბოლოს გაეცნობა გეგმებს, რომელიც სამომავლოდ არის დაგეგმილი ამ სისტემის სრულყოფისკენ.

არსებული მდგომარეობა

დღესდღეობით არსებული მდგომარეობით ქართული ენის ამოცნობა არ წარმოადგენს დიდ სირთულეს, ვინაიდან, თუ გავითვალისწინებთ სხვა სამუშაოებს, რომლებიც ჩასატარებელია იმისთვის, რომ მოხდეს ამ ამოცნობილი სიტყვების, ტექსტის სტრუქტურიზაცია კომპიუტერის მიერ, მივხვდებით რომ ყველაზე მარტივად მოსაგვარებელი და გადასაჭრელი პრობლემა არის ის, რომ კომპიუტერმა ამოიცნოს ქართული სიმბოლოები და მხოლოდ ამის საფუძველზე თქვას, რომ ეს არის ქართული ტექსტი. მსგავს პროდუქტს ინტერნეტში მრავლად შევხვდებით, მათ შორის ისეთი ცნობილი კომპანიების პროდუქტს, როგორიცაა Google-ი, რომელიც იმ შემთხვევაში, თუ გამოვიყენებთ მის სერვისს, გვეტყვის არის თუ არა ტექსტი ქართული, თუმცა მხოლოდ იმის გაგება, რომ ტექსტი ქართულია არაა საკმარისი, საჭიროა ამ ინფორმაციის დამუშავება ისე, რომ მოხდეს მათი სტრუქტურიზაცია, ამოცნობა და შესაბამისად ამის საფუძველზე „გაგება“ იმის, თუ რა შინაარსისაა ტექსტი. საბოლოოდ ეს ყველაფერი დადის ძებნის პრობლემამდე, თუ როგორ რანაირად უნდა მოხდეს ამ ინფორმაციის საფუძველზე მოიძებნოს მისი მსგავსი, შესატყვისი მნიშვნელობა ჩვენს სტრუქტურირებულ ბაზაში, ისე რომ გათვალისწინებული იყოს ქართული ენის სირთულე, მისი უნიკალურიბა და რაც მთავარია მუშაობდეს ზუსტად, რომ არ მივიღოთ დამახინჯებული შინაარსის ტექსტი, რომელსაც აღიქვამს კომპიუტერი.

ისტორიულად არსებობს ინფორმაციული ძებნის ორი მიდგომა: სინტაქსური და სემანტიკური. სინტაქსური ძებნის დროს ძებნის სისტემები უზრუნველყოფენ ძებნას დოკუმენტიდან და მოთხოვნიდან ამოღებული სიტყვებით ან ფრაზებით, შესაბამისად, ასეთი ტიპის ძებნის სისტემების ფუნქციონირება ეფუძნება დოკუმენტისა და მოთხოვნის სინტაქსურ დამთხვევას. ამ პროცესზე უარყოფით გავლენას ახდენს პოლისემია და სინონიმია.

სემანტიკური ძებნა დაფუძნებულია დოკუმენტისა და მოთხოვნის სემანტიკურ შესაბამისობაზე, რომელიც ხორციელდება ბუნებრივი ენის ანალიზის მეთოდებით და განსაზღვრავს დაბრუნებული დოკუმენტების სემანტიკურ მსგავსებას მოთხოვნასთან.

სემანტიკური ძებნის მიდგომები ძირითადად უზრუნველყოფენ სინტაქსური ძებნის მიდგომების გაუმჯობესებას, თუმცა, რიგ შემთხვევებში, სინტაქსური ძებნის გამოყენება ცალსახადაა განსაზღვრული.

სემანტიკური შესაბამისობისათვის აუცილებელია მოთხოვნისა და დოკუმენტის სემანტიკური მნიშვნელობა იყოს ცნობილი. თუ მოთხოვნა განსაზღვრულია ფორმალურად, ყოველი ტერმინის სემანტიკა შეიძლება ცხადად განიმარტოს. თუ მოთხოვნა განსაზღვრულია არაფორმალურად, მაგალითად, ბუნებრივ ენაზე, მაშინ მოთხოვნის ყოველი ტერმინის სემანტიკა უნდა იყოს რაღაცნაირად გახსნილი. პრობლემა იმაშია, რამდენად შეძლებს მანქანა

რომ გაიგოს, რომელი შინაარსობრივი მნიშვნელობა იგულისხმებოდა მოთხოვნაში, რათა მივიღოთ მომხმარებლისათვის აზრობრივად უფრო ახლოს მდგომი და საინტერესო დოკუმენტი.

როგორც ყველა ავტომატიზირებულ სისტემას, საძიებო სისტემასაც გააჩნია გარკვეული პრობლემები, რადგანაც მას უხდება ბუნებრივი ენის თავისებურებათა გათვალიწინება მუშაობის პროცესში. ბუნებრივი ენა აყენებს უამრავი სახის სირთულეს, რომელთა გადაჭრა ძალიან ძნელია თუ სიტყვის კონკრეტული მნიშვნელობა არაა ამოცნობილი. განუსაზღვრელობა მით უფრო ძნელად გადასაჭრელია, თუ სისტემა ტექსტის შინაარსის შემეცნებასთან ერთად, ვერ უზრუნველყოფს რეალური სამყაროს „ადამიანურ“ აღქმას. სემანტიკური ძებნის სისტემას შეუძლია რამოდენიმე ისეთი პრობლემის გადაჭრა, რომელიც ინფორმაციის ძებნისას საკმაოდ ხშირად იჩენს თავს:

➢ სინონიმების პრობლემა.

ხშირად ის, რაც ბუნებრივი ენისათვის „სიმდიდრედ“ ითვლება, ინფორმაციის ძებნის სისტემებს უამრავ პრობლემას უქმნის, მაგალითად, უამრავი სინონიმი. რაც უფრო კარგია „მწერალი“, იმდენი ერთი და იმავე შინაარსის მქონე ტექსტის „კარგად დაწერილი“ ვარიანტები არსებობს. შესაბამისად, საძიებო სისტემას უწევს „გარკვევა“, თუ როგორ გადმოგვცა ავტორმა თავისი აზრი. ამას ემატება ის ფაქტიც, რომ საქართველოს სხვადასხვა კუთხეში ერთი და იმავე ცნებისათვის ზოგჯერ სხვადასხვა ტერმინს იყენებენ. მაგალითად: ჯორკო და სკამი, ბაბუა და პაპა, კომში და ბია და ა.შ. ჩვენ გვჭირდება საძიებო სისტემა, რომელიც დაიჭერს “აზრს“ და არა “სიტყვას“.

➢ პოლისემია.

ქართულ ენაში, ისევე როგორც სხვა ბუნებრივ ენებში, უამრავ სიტყვას გააჩნია ერთზე მეტი მნიშვნელობა, ამასთან კონტექსტში მათ სხვადასხვა აზრი აქვთ, სემანტიკური ძებნის სისტემა შეძლებს მოთხოვნის დამუშავებას შესაბამისი კონტექსტიდან გამომდინარე, რაც პოლისემიის პრობლემატიკის გადაჭრის კარგ საშუალებად გვესახება.

გარდა ზოგადი პრობლემებისა, ცალკე აღსანიშნავია ქართულენოვანი დოკუმენტების ძებნის პრობლემები. მარტივი ანალიზი აჩვენებს, რომ ერთსა და იმავე საძიებო სისტემაში ქართულ და ინგლისურ ენაზე გაკეთებულ მოთხოვნაზე მიღებული შედეგების ხარისხი მკვეთრად განსხვავდება ერთმანეთისაგან. მიზეზად შეიძლება დასახელდეს ქართული ენის მორფოლოგიურ-სინტაქსური სირთულე, რაც არანაკლებ მნიშვნელოვანია; ასევე პრობლემაა ქართული ენის კორპუსების სიმცირე და ხელმისაწვდომობა, რომლებიც, რიგი საძიებო სისტემების მიერ, გამოიყენება ლექსიკონებისა და ონტოლოგიების შესაქმნელად არა მარტო ძებნის, არამედ მანქანური თარგმნის უზრუნველსაყოფად.

მეთოდები და ინსტრუმენტები

ინფორმაციის ძებნის პროცესი არ წარმოადგენს მხოლოდ ერთი სახის ოპერაციის შედეგს. მისი წარმატებულობა და რელევანტურობა დამოკიდებულია ძებნის ციკლის ადეკვატურობაზე და სისრულეზე. ამ ციკლში ერთ-ერთი მნიშვნელოვანი ადგილი უკავია კლასიფიკაციის ეტაპს, რომლითაც, როგორც წესი, იწყება ძებნის პროცესი.

კლასიფიკაცია წარმოადგენს ძებნის პროცესს, რომლის მიზანია ავტომატურად მიანიჭოს კლასები დოკუმეტებს წინასწარ განსაზღვრული სიმრავლიდან. ტექსტების კლასიფიკაცია დამოუკიდებელი ამოცანაა, მაგრამ იგი გამოიყენება საძიებო სისტემებში ძებნის შედეგების გასაუმჯობესებლად.

კლასიფიკაციის ამოცანა ზოგადი სახით შეიძლება ასე ჩამოვაყალიბოთ :

კლასიფიკაციის ამოცანაა მიანიჭოს ბულის მნიშვნელობა წყვილს $\langle dj, ci \rangle \in \mathcal{D}\times C$, სადაც D წარმოადგენს დოკუმენტების დომენს $\mathcal{D} = \{d1, \dots d|D|\}$ და $C = \{c1, \dots c|C|\}$ არის წინასწარ განსაზღვრული კატეგორიების სიმრავლე.

$\langle dj, ci \rangle$ წყვილს მიენიჭება მნიშვნელობა True („1“) თუ იქნება გადაწვეტილება dj მიეკუთვნოს ci კლასს, ხოლო მნიშვნელობა False („0“), არის მაჩვენებელი იმისა რომ, dj არ მიეკუთვნოს ci კლასს.

უფრო ფორმალურად - მიახლოებით განისაზღვროს უცნობი მიზნის ფუნქცია (რომელიც განსაზღვრავს, როგორ უნდა იქნას დოკუმენტი კლასიფიცირებული): Φ:$D\times C \rightarrow \{True$, False$\}$ ისეთი Φ' (Φ' :$D \times C \rightarrow$ {True, False}) კლასიფიკატორის მეშვეობით, რომელიც „მაქსიმალურად თანხვედრილი“ იქნება Φ-სთან.

თანხვედრის/დამთხვევის (რომელსაც „ეფექტურობას უწოდებენ“) შესაფასებლად იყენებენ ინფორმაციული ძებნის ისეთ კლასიკურ სიდიდეებს როგორიცაა: სიზუსტე (precision) და მთლიანობა (recall).

არსებობს დოკუმენტების კლასიფიკაციის მრავალი მეთოდი, თუმცა ყოველი მათგანი ეფუძნება ტექსტის დამუშავების პროცესს. ტექსტის დამუშავება კი, თავის მხრივ, დამოკიდებულია ენის თავისებურებებსა და სირთულეებზე. კლასიფიკაციის ალგორითმები მუშაობენ უკვე რაღაც მეთოდით დამუშავებულ ტექსტებზე. ტექსტის დამუშავების სხვადასხვა გზა შეიძლება არსებობდეს, თუმცა კლასიფიკაციის თითქმის ყველა ცნობილი ალგორითმი იყენებს მის მინიმიზაციას, მასში მხოლოდ ყველაზე ხშირად განმეორებადი სიტყვების დატოვების გზით.

კლასიფიკაციის საწყისი ამოცანაა დოკუმენტის დამუშავების პროცესი, რომელიც მის განსაზღვრული ნორმალიზებული ფორმით წარმოდგენას გულისხმობს. ეს პროცესი რამდენიმე ეტაპად ხორციელდება:

1. **ტექსტის საწყისი დამუშავება**
    - თვისებების ამოღება;
        - ტოკენიზაცია;
        - „სტოპ“ სიტყვების წაშლა
        - სტემინგი, ლემატიზაცია;
    1. ნიშან-თვისებების შერჩევა;
2. **კლასიფიკაციის მეთოდის შერჩევა/ფორმირება.**

კლასიფიკაციის ამოცანებში ინფორმაციული ძებნის ტექნოლოგიები გამოიყენება სამ ფაზაში: ინდექსაცია, კლასიფიკატორის ფორმირება, შეფასების განსაზღვრა.

დამუშავების პირველი ეტაპი ტექსტიდან თვისებების ამოღებაა. ეს ეტაპი მოიცავს ტოკენიზაციას, ანუ ტექსტის ლექსემებად4 დაყოფას. ტექსტის ლექსემებად დაყოფის ყველაზე მარტივი გზა არის ტექსტის გახლეჩა „ჰარის“ პოზიციაში, შედეგად მიიღება სიტყვების სიმრავლე, რომელთაგან ბევრი გამოუყენებელია, ზოგი კი საკმაოდ მნიშვნელოვანია დოკუმენტის კატეგორიის განსაზღვრისათვის. ამიტომ ასეთი „უსარგებლო“ სიტყვები დოკუმენტიდან ამოიღება, დარჩენილი სიტყვები კი მორფოლოგიურად მუშავდება, რაც მოიცავს სტემინგის და ლემატიზაციის პროცესს.

სტემინგი ევრისტიკული პროცესია, რომელიც მოიცავს სიტყვიდან თანდართული აფიქსების5 ჩამოცილებას.

ლემატიზაცია კი სიტყვის მორფოლოგიური ანალიზის დამუშავების სრული ეტაპია, რომლის დროსაც სიტყვიდან ხდება გრამატიკული მნიშვნელობების მქონე დაბოლოებების ჩამოცილება და ბრუნდება ძირითადი, უცვლელი ლექსიკონური ფორმა - ლემა.

როგორც ცნობილია, ზოგიერთი სიტყვა შესაძლოა შეგვხვდეს სხვადასხვა გრამატიკული ფორმით. ლემატიზაციის პროცესის განხორციელების მიზანია სიტყვათა ფორმების წარმოდგენა ერთი კანონიკური ფორმით.

სტემინგისა და ლემატიზაციის პროცესი ინფორმაციის დამუშავების უნიშვნელოვანესი ეტაპია. ძებნის პროცესის გამარტივებისათვის საკმაოდ ეფექტური გამოსავალია დოკუმეტების ავტომატური ანალიზი, გამოუსადეგარ მონაცემთა ნაწილის მოშორება და მხოლოდ სასარგებლო ინფორმაციის დატოვება. ენის თავისებურებებიდან გამომდინარე, სხვადასხვა ბუნებრივი ენებისათვის, ეს პროცესი განსხვავებულია, ორივე პრიცესის მიზანია სიტყვიდან ფუძის მიღება, მაგრამ ამას ახორციელებენ სხვადასხვა გზით: სტემინგი ემყარება წესებს და არ ითვალისწინებს მეტყველების ნაწილებს, ხოლო სიტყვის უცვლელი ნაწილის მიღება მეტყველების ნაწილის კატეგორიის გათვალისწინებით, ლემატიზაციაა.

სტემინგ ალგორითმს ან იგივე სტემერს აქვს სამი დანიშნულება: პირველია სიტყვათა დაჯგუფება თემის მიხედვით. ბევრი სიტყვა წარმოიქმნება ერთი და იმავე ფუძისგან და ერთსა და იმავე არსს გულისხმობს (მაგ. სწავლა, მო-სწავლ-ე, მა-სწავლ-ებელ-ი). ეს სიტყვები ნაწარმოებია პრეფიქს-სუფიქსებით. სტემერები, რომლებიც ინგლისურ ენაზე მუშაობენ, ძირითადად სუფიქსების მოცილებას ანხორციელებენ, რადგან ზოგიერთ შემთხვევაში პრეფიქსები და ინფიქსები სიტყვის მნიშვნელობის შეცვლას იწვევს გამონაკლისები გვხვდება ისეთ ფლექტიურ ენებში, როგორიცაა გერმანული და დანიური , აგრეთვე კონკრეტული სფეროსათვის დამახასიათებელ დოკუმენტებში, მაგ: მედიცინა, ქიმია, სადაც პრეფიქსები და სუფიქსები განსაზღვრავენ სიტყვის არსს. სუფიქსებში გამოიყოფა ორი ძირითადი სახე: “უღლებადი“ წარმონაქმნი, რომლის ძირითადი ფუნქცია გრამატიკული ინფორმაციაა, მოიცავს ინფორმაციას სიტყვის სქესის, რაოდენობის, ხასიათის ან დროის შესახებ. და დერივაციული წარმონაქმნები, რომლებიც არ იწვევენ საწყისი სიტყვის მნიშვნელობის ცვლილებას, ისინი ქმნიან ახალ სიტყვებს უკვე არსებული სიტყვებიდან . წარმოქმნილი სიტყვისგან სუფიქსების მოშორებით მიიღება ფუძე, რომელიც თითქმის იდენტურია მისი მორფოლოგიური ძირისა, რის შემდეგაც თემატურად ერთმანეთთან დაკავშირებული სიტყვები შესაძლებელია განისაზღვროს მათი ფუძეების დამთხვევით.

მეორე დანიშნულება სტემინგ ალგორითმისა უკავშირდება ინფორმაციის ამოცნობის პროცესს, რომელიც ამ ეტაპზე სიტყვიდან ფუძის ამოღებას გულისხმობს. ჩვენ შეგვიძლია დოკუმენტების უკეთ ინდექსაცია ტერმინებით, ხოლო ტერმინები კი სწორედ ფუძის საშუალებით ჯგუფდება.

მესამე დანიშნულება საერთო ფუძის მქონე სიტყვების გაერთიანებაა. ეს, ზოგადად, ამცირებს ლექსიკონის ზომას. პროცესის შედეგად შესაძლებელია მთელი მონაცემების ფუძეებზე დაყვანა, რაც ამცირებს გამოსაყენებელ სივრცეს და ამსუბუქებს სისტემის დატვირთვას. სტემინგის გავრცელებული მიდგომებია:

1. ალგორითმზე დაფუძნებული სტემინგი. ასეთი სტემერები არ ითვალისწინებენ ისეთ ლინგვისტურ დამოკიდებულებას, როგორიცაა სქესი, დრო, ისინი იყენებენ წინასწარ დადგენილ წესებს, რათა გაიგონ მოაშორონ თუ არა აფიქსები. ვინაიდან არაა გათვალისწინებული ლინგვისტური კანონები, შედეგი ხშირად არასწორად აწყობილი სიტყვაა, რომელსაც არ აქვს არანაირი მნიშვნელობა.

2. ლინგვისტიკაზე დამყარებული მიდგომა, სადაც სიტყვის დამუშავება ხდება ლინგვისტური წესებით

სტემინგის ალგორითმები ლიტერატურაში ნახსენები პირველი სტემერი ლოვინსის (J.B.Lovins) ალგორითმია , რომელიც შეიქმნა ჯერ ინგლისური, ხოლო შემდეგ, სხვადასხვა სამეცნიერო კვლევებში, სხვადასხვა ენისათვის განხორციელდა მისი მოდიფიცირება.

ალგორითმი შედგება ორი ეტაპისაგან: სუფიქსების მოშორება და დარჩენილი ძირის დახარისხება. ალგორითი ითვალისწინებს 294 სუფიქსს, რომელიც სიტყვასთან 29 ვარიანტით

გამოიყენება. ამის შემდეგ დარჩენილ ძირს სჭირდება რამდენიმე ისეთი ლინგვისტური პრობლემის მოგვარება, როგორიცაა ორმაგი დაბოლოება (ორი „ლ“ ან ორი „დ“). ამ საფეხურს „ჩაწერის ფაზა“ ეწოდება. ალგორითმში ჩადებული 35 წესი არკვევს, ძირის დარჩენილი ნაწილი მოდიფიცირებული უნდა იყოს, თუ - მთლიანად წაშლილი. ბოლოს, სიტყვათა გაერთიანება ხდება ალგორითმით, რომელიც ეძებს არა ზუსტ, არამედ მსგავს ფუძეებს, რომლებიც მაინც ახლოსაა ერთმანეთთან მნიშვნელობით. ამ ალგორითმით შესაძლებელია ბევრი შეცდომის დაშვება და შემცირება „სიზუსტისა” და „სისრულის”. აღნიშნული პრობლემების თავიდან ასაცილებლად დავსონმა (J.Dawson) შექმნა ლოვინსის სტემერის მოდიფიცირებული ვარიანტი, მის ალგორითმში სუფიქსების ამოღების თავდაპირველი ეტაპი გაუქმებულია და სიტყვები პირდაპირ ჯგუფდება მსგავსი ძირის მიხედვით. ამავე დროს სუფიქსების სია 1200-მდე გაიზარდა.

იმის მიუხედავად, რომ ლოვინსის ალგორითმი ყველაზე ხშირად გამოიყენება, პორტერის სტემერი ყველაზე პოპულარულია ინფორმაციული ძებნის ამოცანებში. იგი აბალანსებს სიმარტივესა და სიზუსტეს. პორტერს აქვს 5 საფეხურიანი ალგორითმი, რომელიც ლექსიკონში არსებულ ყველა სიტყვასთან შეთავსებადია. ალგორითმი ემყარება 60 წესს, რომლებიც შემდეგ იყოფა 5 საფეხურად. ამ ალგორითმის ძირითადი იდეაა, რომ სუფიქსი (ინგლისურ ენაში) თავად შედგება პატარა და მარტივი სუფიქსებისაგან. ისინი სიტყვას ეტაპობრივად (ხუთ ეტაპად) სცილდება. ყოველი წინა ეტაპის დასრულების შემდეგ - ახალი იწყება.

კიდევ ერთი ცნობილი შემოთავაზებაა პაის/ჰასკის (Paice/Husk) სტემერი ინგლისური ენისათვის (43), რომელიც იყენებს 120 წესს სიტყვის ბოლო სიმბოლოსა და თვითონ სიტყვებს შორის დამოკიდებულების დასადგენად. იგი ხორციელდება რამდენიმე ეტაპად. სიტყვის დამუშავების ყველა ეტაპზე ალგორითმი უზრუნველყოფს დაბოლოების ან მოცილებას, ან შეცვლას. თუ აღმოჩნდა პირობა, რომელიც წესს არ შეესაბამება, ალგორითმი წყდება.

გარდა ინგლისური სტემერებისა, ზოგიერთ მკვლევარს აქვს მოდიფიცირებული მიდგომა ან შემოთავაზება სხვა ენებისთვის. ლინგვისტიკაზე დაყრდნობით ენა შესაძლებელია გაიყოს ორ ძირითად კატეგორიად, მათი მორფოლოგიური სტრუქტურის მიხედვით:

ანალიზური ენა და სინთეზური ენა ისეთი ტიპის ენებია, რომლებშიც სიტყვათა შორის გრამატიკული მიმართებები გამოიხატება დამხმარე სიტყვების და ნაწილაკების მეშვეობით, ინტონაციით (ჩინური) ან სიტყვათა თანმიმდევრობით. ანალიზურია ინგლისური, ფრანგული, ახალი სპარსული, ჩინური და ვიეტნამური ენები.

სინთეზურია ენები, რომლებშიც სიტყვათა შორის გრამატიკული მიმართებები გამოიხატება მორფოლოგიური ხერხებით, თვით სიტყვის ფარგლებში აფიქსებისა და ფუძის ფლექსიის (სიტყვათწარმოება) საშუალებით. სინთეზურია ძველი ბერძნული, ლათინური, რუსული, ქართული და სხვა ენები). მეორე კატეგორია კიდევ იყოფა 3 ქვეკატეგორიად : აგლუტინატიური ენა, ფლექტიური ენა, პოლისინთეზური ენა. ეს კლასიფიკაცია იდეალურია, რადგან ყველა ენა სხვადასხვა კატეგორიას ეკუთვნის. შემოღებულია ორი მუდმივი ცვლადი,

სინთეზურობისა და ფლექტიურობის ინდექსი იმისთვის, რომ გვიჩვენოს, რამდენად მიეკუთვნება ერთი ენა რომელიმე კატეგორიას.

სინთეზურობის ინდექსი აღწერს როგორ და რა დონეზე ემატება სიტყვას აფიქსები. ერთი უკიდურესობაა ყველაზე ანალიტიკური ენები, სადაც ყველა მორფემა6 თავისუფალია. ხოლო მეორე - პოლისინთეზური ენაა, სადაც ენას აქვს მიდრეკილება შეიცავდეს წინადადებებს, რომლებიც ერთი სიტყვისგან და სხვადასხვა აფიქსებისგან შედგება.

მეორე ცვლადი, ფლექტიურობის ინდექსი, აღწერს, თუ რამდენად ადვილია სიტყვების მორფემებად დანაწილება. ამ შკალაზე ერთი უკიდურესობაა ენები, სადაც მარტივად ნაწილდება და ნათლად აღიქმევა მორფემები, ხოლო მეორე - მოიცავს სიტყვებს, რომელიც რთულად აღსაქმელი ან გასარჩევი მორფემებისგან შედგება.

ენების სიმრავლე განაპირობებს პრობლემების სიმრავლეს და მათი მოგვარება მხოლოდ არატრადიციული ხერხებითაა შესაძლებელი, რადგან კლასიკური ხერხები ამ შემთხვევაში არ გამოირჩევიან იმ სიზუსტითა და ეფექტურობით, როგორც ინგლისურში ან მის მსგავს ენებში.

ფინური და თურქული ცვალებადი მორფოლოგიის მქონე ენების მაგალითია. თურქულში 23000 ძირია და სიტყვები ფორმირდება მათი გრამატიკული ფუნქციის მიხედვით, სუფიქსების დახმარებით. ეს შედეგი თეორიულად უსასრულო სიტყვებია. სხვადასხვა მიდგომით ცდილობენ ამ პრობლემის შემცირებას, სიზუსტის შენარჩუნებას, მაგალითად „N” გრამების გამოყენებით და აფიქსების მოშორებით, წესებითა და კარგად ცნობილი სუფიქსების სიით, როგორც პორტერის სტემერში.

თუ შევხედავთ ფლექტიურ ენებს, ამ ჯგუფს ძირითადად მიეკუთვნება ინდოევროპული ენები. მათი სტემერის უმრავლესობა დაფუძნებულია პორტერის მიდგომაზე, რადგან ის იდეალურად ჯდება მათ მორფოლოგიურ სტრუქტურაში. ერთი მაგალითია ალგორითმული სტემერი ,,პოპოვიჩი და ვილეტი სლოვენური ენისათვის“, რომელიც იყენებს 5276 კარგად ნაცნობ სუფიქს. ეს ციფრი გაცილებით მაღალია პორტერის ორიგინალურ შემოთავაზებასთან, რადგან სლოვენური მორფოლოგიურად ინგლისურზე გაცილებით მდიდარია. ამ სტემერის საშუალებით ჩატარებული ექსპერიმენტები კარგი შედეგებითა და სიზუსტით დასრულდა. ბევრი მკვლევარი ამტკიცებს, რომ სტემერის ეფექტურობა იზრდება ენის კომპლექსურ სირთულესთან ერთად. მაგ. არაბული, ბერძნული ენებისათვის. ეს უკანასკნელი კიდევ უფრო მეტი სირთულით გამოირჩევა, ამიტომაც ავტორებმა შექმნეს მეტყველების ნაწილებით დაჭდევება (part-of-speech tagging phase), რათა სუფიქსების მოშორებასთან ერთად გაიგონ, სიტყვა რომელ გრამატიკულ კატეგორიას ექვემდებარება. შემდეგ კი, წესისამებრ, შორდება მორიგი სუფიქსები ფრაზის ნაწილებითი დაჭდევების მიხედვით, ამიტომაც ბერძნული სტემერების 96,7% ლექსიკონის ძირს სწორად იღებს.

არსებობს პოლისინთეზური ენები, რომელთათვისაც სტემინგის პროცესი არ განხორციელებულა. მაგალითად, ჩუკოტკურ-კამჩატკური, აფხაზურ-ადიღური და სხვა

მრავალი ენა სამხრეთ-ამერიკულ ენათა ოჯახიდან . სავარაუდოდ, ამ ენებში ის ფაქტი, რომ არსებობს მრავალი მორფემა, ართულებს ტერმინის იდენტიფიკაციის პროცესს.

ამის მიუხედავად, ყველაზე მეტად გამოყენებადი მაინც პორტერის ალგორითმია მისი სიმარტივის, შეგუებადობისა და გაფართოების გამო. ამასთან ერთად, პორტერმა გაამარტივა საქმე და შექმნა „თოვლის გუნდა“ (Snowball) პლატფორმა, სადაც ყალიბდება ახალი სტემერები (48). Snowball, რომელიც წარმოდგენილია ადვილად სასწავლი ენით, იძლევა სტემერისათვის ANSI7 , C ან JAVA ვერსიის გამოყენების შესაძლებლობას. დღეისათვის 25 სხვადასხვა ენაა დამუშავებული Snowball-ის საშუალებით, რომელთაგან უმრავლესობა აგლუტინატიური ან ფლექტიური ენებია.

ქართული ენის ამომცნობი სისტემა

თანამედროვე საინფორმაციო სისტემებში არ არსებობს პროდუქტი, რომელიც მოახდენს ქართული ენის ამოცნობას და კლასიფიკაცია, სწორედ ამიტომ ამ ნაშრომის მთავარი მოტივაცია და მიზანია მოხდეს ქართული ენის დამუშავება, ამოცნობა და მისი კლასიფიკაცია.

ნაშრომში წარმოდგენილია სისტემა, რომელიც ახდენს ქართული ენის ამოცნობას სხვადასხვა საინფორმაციო და სოციალური ქსელიდან (ვიკიპედია, ფეისბუქი, ტვიტერი და სხვა) და ახდენს მის დაშლას და კლასიფიკაციას, ამისათვის სისტემა იყენებს ბაზას, რომელიც ასევე შეიცავს, რამდენიმე ცხრილს და რომელიც აუცილებელია სისტემის გამართულად მუშაობისთვის, ესენია:

- **კავშირების ცხრილი**
- **ნაცვალსახელების ცხრილი**
- **რიცხვითი სახელების ცხრილი**
- **არსებითი სახელების ცხრილი**
- **საერთო ცხრილი**

კავშირების ცხრილი

ვინაიდან ქართულ ენაში კავშირების სასრული სიმრავლე გვაქვს და არ წარმოადგენს დიდ რაოდენობას, ამ ცხრილი შევსება მოხდა ხელით და შეიცავს მხოლოდ და მხოლოდ კავშირებს, როგორიცაა : და, ან ...

ნაცვალსახელების ცხრილი

ნაცვალსახელებიც ქართულ ენაში არ წარმოდაგენს დიდ სიმრავლს და მისი შევსებაც მოხდა ხელით.

რიცხვითი სახელების ცხრილი

ვინაიდან რიცხვითი სახელი უსასრულო რაოდენობის შეიძლება იყოს ამიტომ ამ შემთხვევაში ბაზაში წერია მხოლოდ საწყისი ფორმები რიცხვითი სახელის რომლისგანაც შემდეგ იწარმოება სხვა რიცხვითი სახელები.

მაგალითად. „ერთი“ , „ორი“ ...

არსებითი სახელების ცხრილი

ვინაიდან არსებითი სახელები ძალიან დიდი რაოდენობითაა ამ შემთხვევაში ამ ცხრილში არის 8000-მდე არსებითი სახელი საწყისი ფორმით.

მაგალითად. „სახლი“ და არა „სახლმა“ ან „სახლს“.

საერთო ცხრილი

ამ ცხრილს პროგრამა ავტომატურად შეავსებ და ჩაწერს ყველა იმ სიტყვას, რომელიც ამოცნობილი იქნება, როგორც ქართული სიტყვა იმ ფორმით, რა ფორმითაც მოცემული იქნება ტექსტში.

ამ ცხრილებზე დაყრდნობით ნაშრომში წარმოდგენილი სისტემის მუშაობის პრინციპი შემდეგია:

1. შეგვყავს ჩვენთვის საინტერესო ტექსტი ან ვეძებთ მას სოციალურ ქსელში.
2. პროგრამა ახდენს შეყვანილი ტექსტის იდენთიფიცირებას არის თუ არა ტექსტი ქართული
3. თუ კი, მაშინ პროგრამა გადადის მე-4 ბიჯზე, თუ არა, აბრუნებს რომ აღნიშნული ტექსტი არ არის ქართული.
4. ამ ეტაპზე ხდება დაბრუნებული ტექსტის დაშლა ცალკეულ სიტყვებად და მათი ჩაწერა მასივში, რომ მოხდეს მათი დამუშავება.
5. იშლება მასივიდან **კავშირები, რიცხვითი სახელები, ნაცვალსახელები.**
6. ხდება არსებითი სახელების ამოცნობა და გამოტანა ცალკე ცხრილში.

ბიჯი #1

დავუშვათ სისტემისთვის გადაცამული ტექსტი არის შემდეგი:

„უძველეს ხანაში ადამიანები მშვიდობიანად ცხოვრობდნენ ერთად, არც ქალაქები ჰქონიათ, არც კანონები, არც რამე წეს-წყობილება; ლაპარაკობდნენ საერთო ენით და ემორჩილებოდნენ ერთ ღმერთს-ზევს. დროთა ვითარებაში ჰერმსმა სხვადასხვა კილოკავი გააჩინა და კაცობრიობა გაყო ხალხებად.“

გავნიხილოთ თითოეული ბიჯი აღნიშნულ მაგალითზე.

ბიჯი #2

მეორე ბიჯზე პროგრამა მოახდენს ამოცნობას არის თუ არა ტექსტი ქართული. პირველ ეტამპზე პროგრამა სიმბოლოების დონეზე ადარებს მოცემულ ტექსტს ქართული ენის სიმბოლოებთან „ა-ჰ“ შუალედში, თუ აღნიშნული ტექსტი შეიცავს ერთს მაინც ასეთ სიტყვას, რომელიც შეიცავს ქართულ სიმბოლოებს, პროგრამა გვეტყვის, რომ შეყვანილ ტექსტში არის ქართული სიმბოლოები და გადადის მესამე ბიჯზე.

მაგალითად: პროგრამა აიღებს პირველ სიტყვას **„უძველეს“** დაშლის მას სიმბოლოებად -> „უ“,„ძ“ ,“ვ“,“ე“,“ლ“,“ე“,“ს“,“ი“ და შეამოწმებს არის თუ არა თითოეული სიმბოლო „ა-ჰ“ შუალედში და ასე გააგრძელებს ბოლო სიტყვამდე.

ბიჯი #3

ტექსტი შეიცავს ერთს მაინც ქართულ სიტყვას, ამიტომ პროგრამა გადადის შემდეგ ბიჯზე

ბიჯი #4

მეოთხე ბიჯზე აღნიშნული ტექსტის დაშლა ხდება ცალკეულ სიტყვებად. აღნიშნული ტექსტის მაგალითზე, რომ განვიხილოთ მივიღებთ დაახლოებიტ ასეთ შედეგს, როგორც მოცემულია ქვევით

Array = { „*უძველეს*“, „*ხანაში*“, „*ადამიანები*“, „*მშვიდობიანად*“ ... }

ბიჯი #5

მეხუთე ბიჯი წარმოადგენს უფრო რთულ პროცესს, ვიდრე წინა დანარჩენი ბიჯები იყო, ვინაიდან აქ ხდება ტექსტის ამოცნობის და დამუშავების პირველი ეტაპები.

კავშირების ამოცნობა - ვინაიდან გვაქვს ცხრილი კავშირების სასრული რაოდენობით, კავშირების ამოცნობა და ამოშლა ტექსტიდან არ წარმოადგენს სირთულეს, ვინაიდან მეოთხე ბიჯის შემდეგ მიღებული მასივის თითოეული ელემენტის შემოწმება ხდება კავშირების ცხრილთან და თუ მსგავსი მოიძებნება ცხრილში, მაშინ იშლება ეს სიტყვა მასივიდან.

მაგალითად. „არც“ და „და“ ამოიშლება მასიიდან.

ნაცვალსახელების ამოცნობა - მსგავსად კავშირებისა თუ მოხდება ბაზაში არსებული მნიშვნელობის და ტექსტში არსებული მნიშვნელობის დამთხვევა იშლება მასივიდან.

რიცხვითი სახელის ამოცნობა - ამ შემთხვევაში ყველაფერი ისე ხდება როგორც ზემოთ განხილულ ორ შემთხვევაში.

მაგალითად. *„ერთი“, „ორი“ მსგავსი სიტყვები ამოიშლება მასივიდან*

თუმცა არსებობს „რთული“ რიცხვითი სახელები, მაგალითად, როგორიცაა ათას ორას ოცდახუთი. ამ შემთხვევაში ბაზაში მსგავსი სიტყვა არ გვაქვს და გვიწევს არსებულ სიტყვებზე მანიპულაციების ჩატარება.

მაგალითად. რიცხვითი სახელი არ გვაქვს საწყისი ფორმით მოცემული მაშინ პროგრამა ადარებს გადაცემულ სიტყვას, მაგალითად ამ შემთხვევაში „ერთ“ ბაზაში არსებულ სხვა სიტყვებს, როგორიცაა ერთი, ორი , სამი, ოთხი ..., ვინაიდან დამთხვევა არ მოხდა პროგრამა შეადარებს, თუ შეიცავს, რომელიმე ბაზაში არსებული სიტყვა ქვესტრიქონდა „ერთ“-ს და თუ შეიცავს, მაშინ ეს სიტყვა რიცხვითი სახელია, ამ შემთხვევაში „ერთ“ ქვესტრიქონია „ერთი“ -ის.

გარდა ამისა არსებობს რიგობითი და წილობითი რიცხვითი სახელი, რომელთა საწყისი პირველი ორი ასო არის „მე“ და ბოლო ასო „დი“ ან „ე“ , პროგრამა ასევე აკეთებს მათ ამოცნობას და ამოშლას.

მაგალითად. „მეთხუთმეტედი“, „მეორე“

ბიჯი #6

ყველაზე რთული პროცესი ამ სისტემის მუშაობაში არის არსებითი სახელების ამოცნობა, ვინაიდან თუ გავითვალისწინებთ ქართული ენის სირთულეს, რთულია ამოიცნო არსებითი სახელები სხვადასხვა ფორმაში არსებითი სახელების ბაზის გარეშე, რომელ ბაზაშიც ჩაწერილი იქნება არსებითი სახელები ყველა ფორმაში, თან ისე რომ დაადგინო ფუძე და გაითვალისწინო კუმშვა კვეცის პრობლემა.

ცხადია თუ იარსებებს არსებითი სახელების მსგავსი ბაზა, რომელშიც ყველა ფორმაში ჩაწერილი არსებითი სახლები იქნება ეს პრობლემაც მოგვარდება და ადვილად მოხდება მათი ამოცნობა და კლასიფიკაცია, თუმცა მსგავსი ბაზის შექმნა წარმოუდგენლად მიმაჩნია და სწორედ ამიტომამ ნაშრომში წარმოდგენილია მეთოდი, რომელიც გვერდს აუვლის კუმშვა-კვეცის პრობლემას, გაიგებს ფუძეს და დაითვლის ფუძის მიხედვით სიტყვათა სიხშირეებს.

იმისათვის რომ გვერდი აუაროთ კუმშვა-კვეცას ამისათვის თითოეული სიტყვიდან ხდება ხმოვნების ამოშლა.

მაგალითად. თუ გვაქვს სიტყვა „სახლს“ ან „ადამინმა“ -> დაგვრჩება „სხლ“ და „დმნმ“

ამის შემდეგ „სხლ“ -ს ძებნა ხდება არსებითი სახელების. ამ შემთხვევაში ბაზაში დიდი ალბათობით იარსებებს ისეთი სიტყვები, რომლებიც იმის შემდეგ რაც ხმოვნებს მოვაშორებთ დარჩება „სხლ“, მაგალითად ასეთი სიტყვაა „სხეული“, შესაბამისად პროგრამამ ბაზიდან დაგვიბრუნა ორი მნიშვნელობა

1. სახლი
2. სხეული

ამის შემდეგ დგება მეორე პრობლემა როგორ უნდა ავარჩიოთ, რომელია ამ სიტყვის საწყისი მნიშვნელობა?!

ამისათვის პირველ რიგში ვამოწმებსთ გადაცემული ტექსტის სიგრძე მეტია თუ არა ბაზაში არსებულ მნიშვნელობებზე, ვინაიდან ბრუნვაში არსებული სიტყვა არ შეიძლება სიმბოლოების რაოდენობით ნაკლები იყოს საწყის ფორმაში მოცემულ სიტყვაზე.

მაგალითად. „სახლს“ -> შეიცავს 5 სიმბოლოს, ხოლო „სხეული“ -> 6 სიმბოლოს, რაც ავტომატურად გამორიცხავს, იმას რომ ეს სიტყვა შეიძლება იყოს გადაცემული სიტყვის საწყისი მნიშვნელობა.

გარდა ზემოთ აღნიშნულია ხდება თითოეული სიმბოლოს სიმბოლოზე შემოწმება.

მაგალითად. თუ გვაქვს ბაზაში ასევე „სახელი“ ->“სხლ“ -> 5 , ეს სიტყვა ანალოგია „სახლს“-ის ამ შემთხვევაში შემოწმებას ექნბა შემდეგი სახე

ს	ა	ხ	ლ	ს	
ს	ა	ხ	ე	ლ	ი
+	+	+	-	-	-

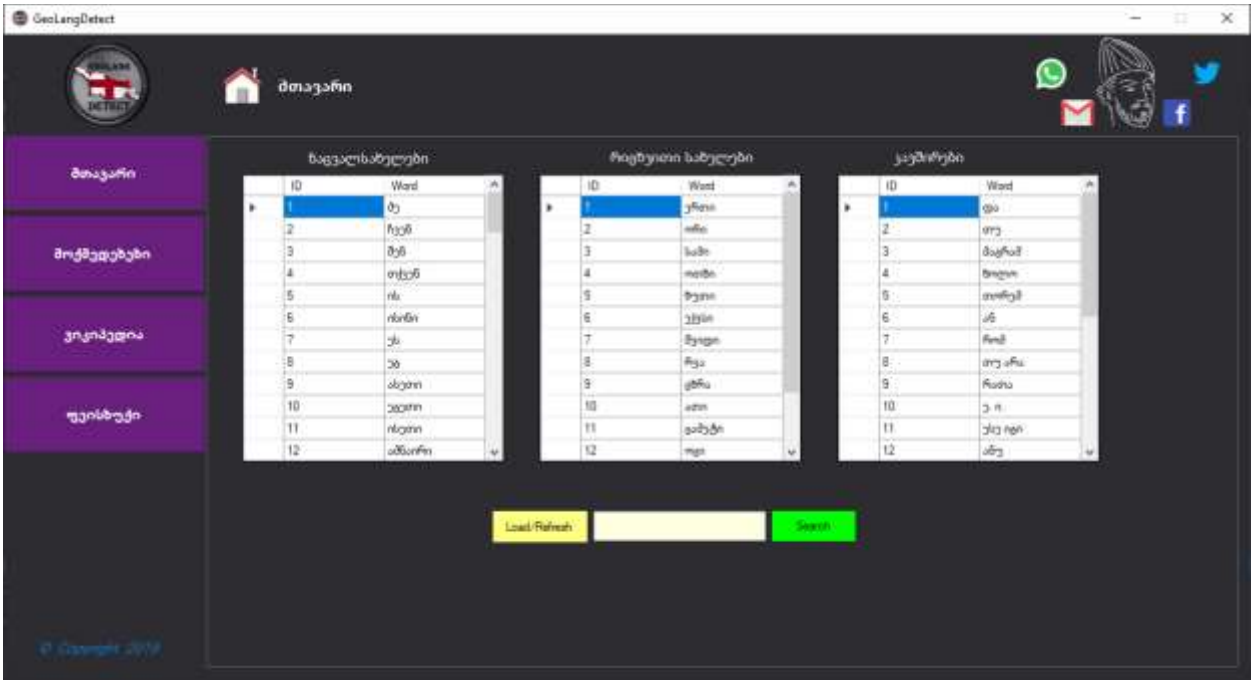
განსხვავება სიმბოლოებს შორის მეტია 1-ზე და შესაბამისად ესეც გამორიცხავს სიტყვის საწყის მნიშვნელობას და სწორედ ამ მეთოდებით მივყვებით ბოლომდე, სანამ არ ვიპოვით რეალურ მნიშვნელობას, თუმცა არ დაგვავიწყდეს, რომ აუცილებებლია არსებითი სახელების სრულყოფილი საწყისი ფორმების ბაზა.

ეს არის ნაშრომში წარმოდგენილი ქართული ენის ამომცნობი სისტემის მუშაობის პრინციპი, რომელსაც შემდგომში აუცილებლად დაემატება კლასიფიკაციის მეთოდი, რომელიც მხოლოდ და მხოლოდ დამოკიდებულია შესაბამისი მიმართულების ბაზების არსებობაზე.

სისტემის პრაქტიკული რეალიზაცია აღწერილია შემდეგ თავში და და პროგრამული კოდები მოცემულია დანართის სახით.

ამოცანის პრაქტიკული რეალიზაცია

ამოცანის პრაქტიკული რეალიზაციისთვის გამოყენებულია Microsoft SQL Server-ი, Visual Studio და ობიექტზე ორიენტირებული პროგრამირების ენა C#, რომელის საშუალებითაც შეიქმნა Windows Form აპლიკაცია, რომელიც მოცეულია **სურათ 1**-ზე.

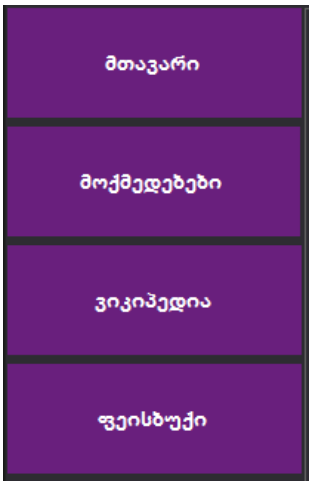


სურათი 1. ბაზაში არსებული სიტყვები სიხშირეებით.

პროგრამა შედგება სამი ნაწილისგან ესენია :

7. Header - სადაც განთავსებულია ლოგო და მაჩვენებელი იმისა, თუ რომელ გვერძე ვდგავართ, ასევე კონტაქტისთვის აუცილებელი ყველაზე ხშირად გამოყენებადი სისტემები.

8. Menu - აქ განთავსებულია ღილაკები, რომელთა საშუალებითაც ადვილად ხდება ნავიგაცია ერთი გვერდიდან მეორეზე.



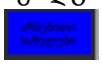
9. MainPanel- სადაც ხდება ძირითადი მოქმედებები იმის შესაბამისად, თუ რომელ გვერძე ვდგავართ.

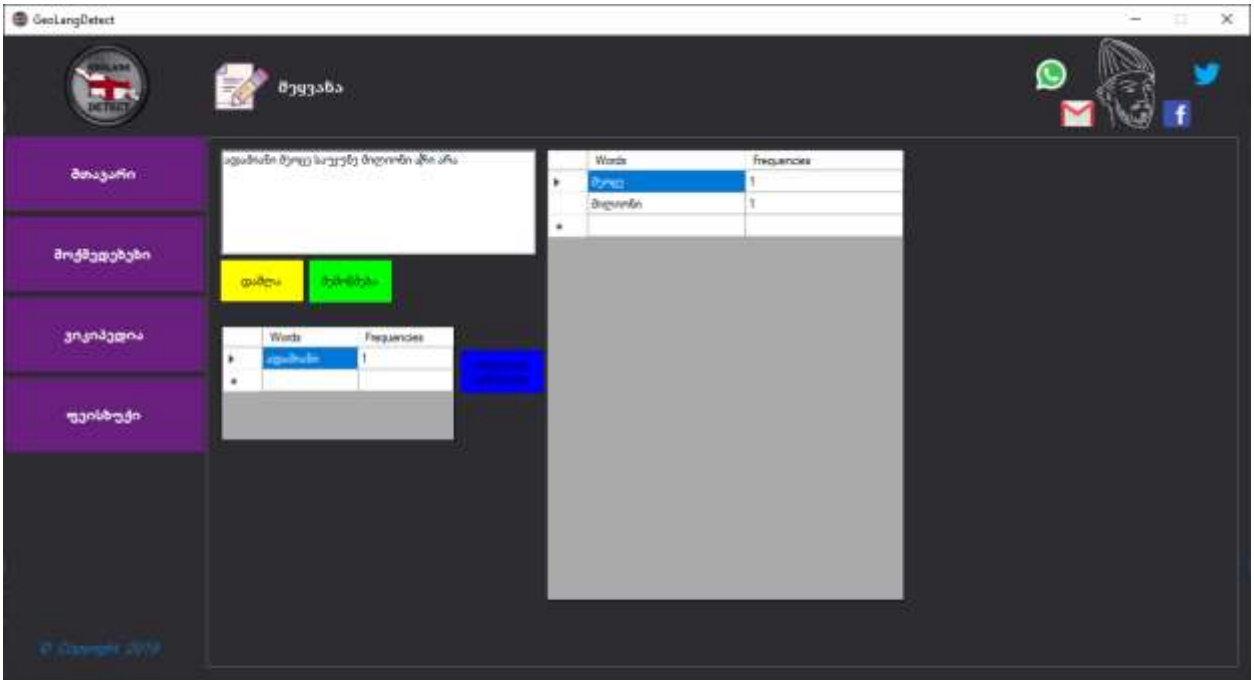
მთავარ გვერდზე განთავსებულია ცხრილები **(იხ. სურათ 1**) , რომელიც გვიჩვენებს, თუ რომელი ცხრილების და მათში განთავსებული მნიშვნელობების გამოყენება უწევს პროგრამას, იმისათვის რომ ამოიცნოს კავშირები, ნაცვალსახელები, რიცხვითი მნიშვნელობები და არსებითი სახელები. აღნიშნული ცხრილების შევსება ხდება Microsoft SQL Server-დან, სადაც წინასწარ არის გაწერილი აღნიშნული ბაზები.

- ✓ **Load/Refresh**  ღილაკით ხდება ბაზების განახლება და ხელახლა ჩატვირთვა,

- ✓ **Search**  ღილაკით კი შეყვანილი მნიშვნელობის ძებნა.

მოქმედებების გვერძე (**იხ. სურათი 2**) გვაქვს სტანდარტული Textbox, სადაც ხდება ჩვენს მიერ ნებისმიერი სახის ტექსტის შეყვანა, რომლის დამუშავებაც გვინდა.

- ✓ **შემოწმება**  - ღილაკი ახდენს შეყვანილი ტექსტის შემოწმებას იმაზე, არის თუ არა ეს ტექსტი ქართული.
- ✓ **დაშლა**  – ახდენს აღნიშნული ტექსტიდან კავშირების, ნაცვალსახელების, რიცხვითი სახელების ამოღებას და დანაწევრებასც ცალკეულ სიტყვებად, რომელი სიტყვებიც შემდგომ გამოდის ცხრილის სახით.
- ✓ **არსებითი სახელები**  - ახდენს ტექსტში არსებული არსებითი სახელების ამოღებას და გამოტანას ცალკე ცხრილში.



სურათი 2. ტექსტის შეყვანა

ვიკიპედია გვერდზე (**იხ. სურათი 3**) ვიკიპედიიდან წამოღებული ინფორმაცია, რომლის შემოწმებაც ხდება შეიცავს, თუ არ ქართულ სიტყვებს, თუ კი ამის შემდეგ ხდება ამ ინფორმაციის დაშლა სიტყვებად, გაფილტრვრა და მოთავსება დროებით ცხრილში, სადაც თუ რომელიმე არასწორი სიტყვა მოხვდება შეგვიძლია ხელით მოვახდინოთ მათი ამოშლა და იმის შემდეგ რაც დავრწმუნდებით, რომ სიტყვები ნამდვილად შეესაბამება სწორ ფორმებს ვამატებთ მათ ბაზაში თავისი სიხშირეებით.

- ✓ **ძებნა**  - ამ ღილაკით ხდება საძიებო სიტყვის ძებნა ვიკიპედიაში და გამოტანა შესაბამისი კონტენტის.

- ✓ **შემოწმება**  - ახდენს შემოწმებას შეიცავს თუ არა ქართულ სიტყვებს.

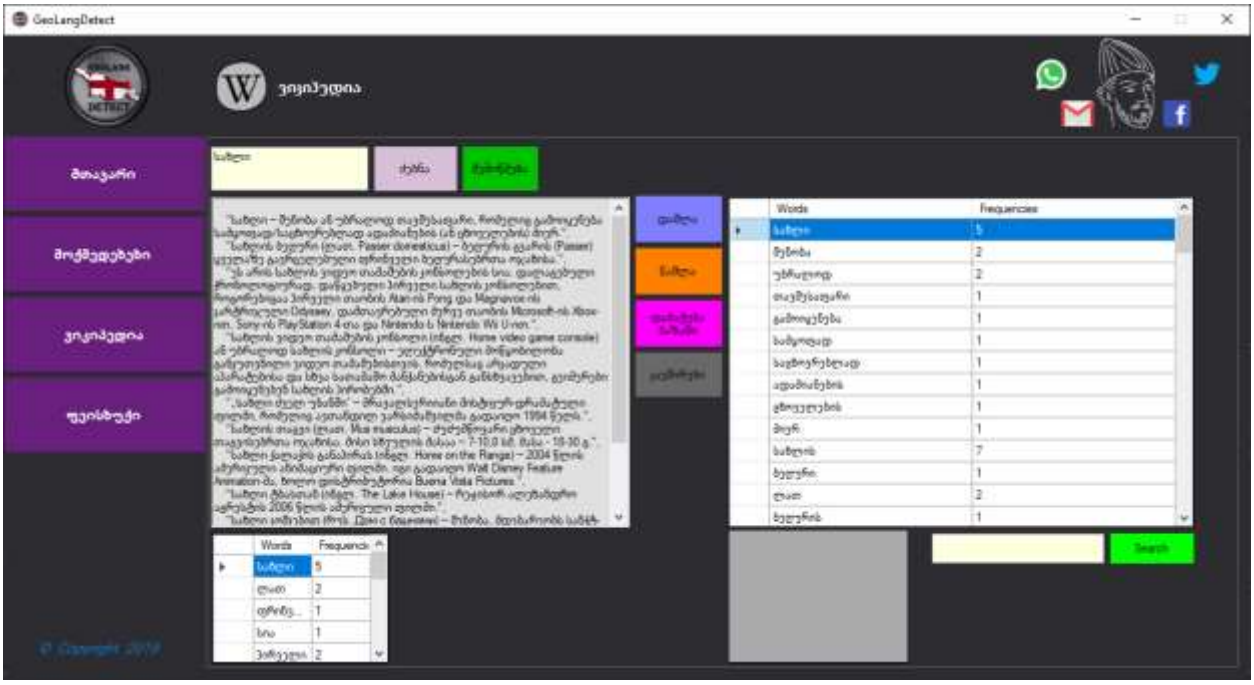
- ✓ **დაშლა**  - ახდენს დაბრუნებული ტექსტის დაშლას ნებისმიერ სიმბოლოზე, ამის შემდგომ იღებს კავშირებ, ნაცვალსახელებს, არსებით სახელებს და რიცხვით სახელებს და დარჩენილი სიტყვების მასივს გვიბრუნებს ცხრილის სახლით.

- ✓ **წაშლა**  - იმ შემთხვევაში თუ რომელიმე სიტყვა არასწორია ან შეცდომით მოხვდა ცხრილში, შეგვიძლია ეს სიტყვა ან სიტყვენი ამოვშალოთ ცხრილიდან, რადგან არ მოხდეს ბაზის „დაბინძურება“.

- ✓ **დამატება ბაზაში**  - ახდენს ქართული სიტყვების დამატებას მონაცებთა საერთო ბაზაში.

- ✓ **კავშირები**  - გამოაქვს მოცემულ ტექსტში არსებული კავშირები ცალკე ცხრილის სახით.

- ✓ **Search**  - ახდენს ბაზაში მონაცემის ძებნას.



სურათი 3. ვიკიპედიიდან წამოღებული ინფორმაციის დამუშავება

რაც შეეხება მეოთხე გვერდს **ფეისბუქი,** ის განიხილება მომავლის პერსპექტივად, სადაც მოხდება ფეისბუქის პოსტების ამოღება და ამ ინფორმაციის დამუშავება იმ სახით, რა სახითაც ხდება ვიკიპედიის შემთხვევაში.

დასკვნა

მოცემული ნაშრომის მიზანია მოახდინოს ქართული ენის პოპულარიზაცია და ამასთანავე ქართული ენის „გათანამედროვება“, რაც გულისხმობს იმას რომ ქართული ენა, ადვილად აღქმადი გახდეს თანამედროვე კომპიუტერული სისტემებისთვის.

ყველაზე რთული პროცესი ქართული ენის კლასიფიკაციაა, რომელიც სრულიად ახალი მიდგომების საფუძველზე დაძლეულია და საჭიროებს დახვეწას და განვითარებას. ახალ მიდგომებში იგულისხმება კუმშვა-კვეცის თავიდან აცილება ხმოვნების ამოშლით სიტყვიდან და შემდეგ მოქმედებები ამ სიტყვაზე რამაც გარკვეული შედეგები დადო და მოახდინა ტექსტის კლასიფიკაცია სრულყოფილად არა მაგრამ მაღალი სიზუსტით.

აქედან გამომდინარე ამ ნაშრომში განხორციელებული სამუშაო შესაძლოა გახდეს საფუძველი, იმისა რომ შეიქმნას და აიგოს სრულფასოვანი სისტემა ქართული ენის კლასიფიკაციისათვის, რომელიც ასევე საფუძველი იქნება, იმისა რომ შეიქმნას პირველი ქართული საძიებო სისტემა.

გამოყენებული ლიტერატურა

[1]მ. არჩუაძე, "ქართულენოვანი ტექსტების კლასიფიკაციის ამოცანა ინფორმაციულ ძებნაში", პროფესორი, თბილისის სახელმწიფო უნივერსიტეტი, 2017.

[2]T. Korenius and K. Järvelin, Stemming and Lemmatization in the Clustering of Finnish Text Documents. DBLP, 2004.

[3]M. Ali Dootio and A. Imdad Wagan, AUTOMATIC STEMMING AND LEMMATIZATION PROCESS FOR SINDHI TEXT. 2017.

[4]C. D. Manning, P. Raghavan and H. Schütze, Introduction to Information Retrieval. Stanford University, 2008.

[5]A. Karl Ingason, S. Helgadótti, H. Loftsson and E. Rögnvaldsson, A Mixed Method Lemmatization Algorithm Using a Hierarchy of Linguistic Identities. Reykjavik, 2014.

[6]C. Luca and A. Fatima, New Framework for Semantic Search Engine. Cambridge, 2014.

[7]H. Bast, B. Buchhold and E. Haussmann, Semantic Search on Text and Knowledge Bases. 2016.

დანართი

#1

```
//ძებნა ვიკიპედიაში
        public void WikipediaSearch (String valueToSearch)
        {
            WebClient client = new WebClient();
            var a =
client.DownloadString("https://ka.wikipedia.org/w/api.php?action=opensearch&search=" +
TxtWikiSearch.Text);
            var b = Newtonsoft.Json.JsonConvert.DeserializeObject(a);
            string[] c = b.ToString().Split('[');
            var i = c[3];
            //MessageBox.Show(i);
            //for (int i = 0; i < c.Length; i++)
            //{
                TxtWikiOutput.Text = i;
            //}

        }
```

#2

```
//ტექსტის დაშლა ცალკეულ სიტყვებათ და მათი სიხშირეების დათვლა

public NameValueCollection SplitText(String text)
        {
            NameValueCollection list = new NameValueCollection();
            string[] array = text.Split(' ','\t', '\n','.','?','"','!','%','(',')','-
',',',']','_','–','/');
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i].Trim()=="")
                {
                    array = array.Where(w => w != array[i]).ToArray();
                }
            }
            int n = 1;
            string[] a;
            int b;
            for (int i = 0; i < array.Length; i++)
            {
                if (list[array[i]] != null)
                {
                    a = list.GetValues(array[i]);
                    b = Int32.Parse(a[0]) + 1;
                    list.Set(array[i], b.ToString());
                }
                else
                {
                    list.Add(array[i], n.ToString());
                }
            }
            return list;
```

```
        }
```

#3

```
//სიტყვების  შემოწმება შეიცავს თუ არა ქართულ სიტყვებს
        public bool CheckGeoWord(String str)
        {
            int check = str.Length;
            int counter = 0;
            Char first = 'ა';
            Char last = 'ჰ';
            if (str.Length == 1)
            {
                return false;
            }
            else
            {
                foreach (char ch in str.ToCharArray())
                {
                    if ((int)ch >= (int)first && (int)ch <= (int)last)
                    {
                        counter++;
                    }
                    else
                    {
                        continue;
                    }
                }
                if (counter == str.Length)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
```

#4

```
//სიტყვების  შემოწმება  მონაცემთა ბაზაში ბაზაში
        public bool ChechGeoWordDb(NameValueCollection value)
        {
            con.Open();
            int allWord = 0;
            int inDb = 0;
            foreach (string s in value)
            {
                foreach (string v in value.GetValues(s))
                {
                    if (CheckGeoWord(s))
                    {
                        var check_User_Name = new SqlCommand("SELECT
[ID],[Word],[Frequency] FROM[GeoLang].[dbo].[Words]WHERE Word Like N'" + s + "'", con);
```

```
                        if (check_User_Name.ExecuteScalar() != null)
                        {
                            int UserExist = (int)check_User_Name.ExecuteScalar();
                            if (UserExist > 0)
                            {
                                inDb++;
                                allWord++;
                            }
                        }
                        else
                        {
                            allWord++;
                        }
                    }
                    else
                    {
                        allWord++;
                        continue;
                    }
                }
            }
            if (inDb>0)
            {
                con.Close();
                return true;
            }
            else
            {
                con.Close();
                return false;
            }

        }
```

#5

```
//აბრუნებს მასივს კავშირების გარეშე
        public NameValueCollection CheckKavshirebi (NameValueCollection value)
        {
            NameValueCollection value2 = new NameValueCollection();
            con.Open();
            foreach (string s in value)
            {
                foreach (string v in value.GetValues(s))
                {
                    if (CheckGeoWord(s))
                    {
                        var check_User_Name = new SqlCommand("SELECT[Word] FROM
[GeoLang].[dbo].[Kavshirebi] WHERE Word Like N'" + s + "'", con);
                        if (check_User_Name.ExecuteScalar() != null)
                        {
                            string UserExist =
check_User_Name.ExecuteScalar().ToString();
                            if (UserExist == s)
                            {
                                //value2.Add(s,v);
```

```
                            continue;
                        }
                    }
                    else
                    {
                        value2.Add(s, v);
                        //continue;
                    }
                }
                else
                {
                    value2.Add(s, v);
                    //continue;
                }
            }
        }
        con.Close();
        return value2;
    }
```

#6

```
//აბრუნებს მასივს ნაცვალსახელების გარეშე გარეშე
public NameValueCollection CheckNatsvalsaxeli (NameValueCollection value)
    {
        NameValueCollection value2 = new NameValueCollection();
        con.Open();
        foreach (string s in value)
        {
            foreach (string v in value.GetValues(s))
            {
                if (CheckGeoWord(s))
                {
                    var check_User_Name = new SqlCommand("SELECT[Word] FROM
[GeoLang].[dbo].[NacvalSaxeli] WHERE Word Like N'" + s + "'", con);
                    if (check_User_Name.ExecuteScalar() != null)
                    {
                        string UserExist =
check_User_Name.ExecuteScalar().ToString();
                        if (UserExist == s)
                        {
                            //value2.Add(s,v);
                            continue;
                        }
                    }
                    else
                    {
                        value2.Add(s, v);
                        //continue;
                    }
                }
                else
                {
                    value2.Add(s, v);
                    //continue;
```

```
                }
            }
        }
        con.Close();
        return value2;
    }
```

#7

```
//ამოწმებს  რიცხვით სახელებს მონაცემთა საწყის ბაზაში ბაზაში
    public bool checkNumberOriginal(string checkstr)
    {
        for (int i = 0; i < number.Length; i++)
        {
            if (checkstr == number[i])
            {
                return true;
            }
            else
                continue;
        }
        return false;
    }
```

#8

```
//ამოწმებს რიგორბით რიცხვით სახელებს

public bool checkRigobiti(string checkstr)
    {
        for (int i = 0; i < number.Length; i++)
        {
            if (checkstr.Contains("მე") && checkstr[checkstr.Length - 1] == 'ე' &&
checkstr.Contains(number[i].Remove(number[i].Length - 1)))
            {
                return true;
            }
            else
                continue;
        }
        return false;
    }
```

#9

```
//ამოწმებს წილობით რიცხვით სახელებს

public bool checkWilobiti(string checkstr)
        {
            for (int i = 0; i < number.Length; i++)
            {
                if (checkstr.Contains("მე") && checkstr[checkstr.Length - 1] == 'ი' &&
checkstr[checkstr.Length - 2] == 'დ' &&
checkstr.Contains(number[i].Remove(number[i].Length - 1)))
                {
                    return true;
                }
                else
                    continue;
            }
            return false;
        }
```

#10

```
//აბრუნებს მასივს რომელშიდაც წერია რიცხვითი სახელები

public NameValueCollection checkNumber(NameValueCollection checkstr)
        {
            NameValueCollection newColection = new NameValueCollection();
            NameValueCollection wordColection = new NameValueCollection();
            //რვა და ცხრა თუა აღარ ჩამოაჭრის ბოლო სიმბოლოს და პირდაპირ შეამოწმებს
            for (int i = 0; i < number.Length; i++)
            {
                foreach (string s in checkstr)
                {
                    foreach (string v in checkstr.GetValues(s))
                    {
                        if (s.Contains("რვა") || s.Contains("ცხრა") ||
checkNumberOriginal(s))
                        {
                            if (checkValueInCollection(newColection, s))
                            {
                                break;
                            }
                            else
                                newColection.Add(s, v);
                            //newColection.Add(s, v);
                        }
                        else if (checkWilobiti(s))
                        {
                            if (checkValueInCollection(newColection, s))
                            {
                                break;
                            }
                            else
                                newColection.Add(s, v);
```

```
                    //newColection.Add(s, v);
                }
                else if (checkRigobiti(s))
                {
                    if (checkValueInCollection(newColection, s))
                    {
                        break;
                    }
                    else
                        newColection.Add(s, v);
                    //newColection.Add(s, v);
                }
                else if (s.Contains(number[i].Remove(number[i].Length - 1)))
                {
                    int counter = 0;
                    for (int j = 0; j < number.Length; j++)
                    {
                        if (s.Contains(number[j].Remove(number[j].Length - 1)))
                        {
                            counter++;
                        }
                        else
                        {
                            continue;
                        }
                    }
                    if (counter >= 2)
                    {
                        if (checkValueInCollection(newColection, s))
                        {
                            break;
                        }
                        else
                            newColection.Add(s, v);
                        //newColection.Add(s, v);
                    }
                    else
                    {
                        break;
                    }

                }
                else
                {
                    string str = wordColection[s];
                    if (str != null)
                    {
                        continue;
                    }
                    else
                        wordColection.Add(s, v);
                }
            }
        }
    }
    return newColection;
}
```

#11

```
//ხმოვნების ამოშლა სიტყვიდან
        public string removeWords (string str)
        {
            var charsToRemove = new string[] { "ა", "ე", "ი", "ო", "უ" };
            foreach (var c in charsToRemove)
            {
                str = str.Replace(c, string.Empty);
            }
            return str;
        }
```

#12

```
//ორი სტრინგის არსებითი სახელის შედარება ბაზაში არსებულ არსებით სახელთან
        public bool compareTwoString (string str, string str2)
        {
            int[] arr = new int[100];
            int i = 0;
            int counter = 0;
            foreach (var item in str)
            {
                foreach (var item2 in str2)
                {
                    if (item==item2)
                    {
                        arr[i] = 1;
                        i++;
                    }
                    else
                    {
                        arr[i] = 0;
                        i++;
                    }
                }
            }
            for (int j = 0; j < arr.Length; j++)
            {
                counter += arr[j];
            }
            if (counter>str2.Length-1)
            {
                return true;
            }
            else
                return false;
        }
```

#13

```
//აქ ხდება ბაზაში არსებულ არსებით სახელებთან შედარება გადაცემული ტექსტის/სიტყვის
        public bool checkArsebitiSaxeli (string str)
        {
            if (con.State!=ConnectionState.Open)
            {
                con.Open();
            }
            SqlDataReader reader;
            SqlDataReader reader2;
            string str2 = removeWords(str);
            var check_arsebiti = new SqlCommand("SELECT DISTINCT [Word] [SubWord]
FROM[GeoLang].[dbo].[Arsebiti] Where SubWord like N'" + str2 + "'", con);
            reader = check_arsebiti.ExecuteReader();
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    //იმ შემთხვევაში თუ ეს ორი სიტყვა შეიცავს ერთნაირ სიმბოლოებს
                    //და ამასთანავე გადაცემული სიტყვის სიგრძე მეტია ან ტოლი ბაზაში
არსებულზე მაშინ ეს სიტყვაა
                    if (compareTwoString(str, reader.GetValue(0).ToString()) &&
str.Length>=reader.GetValue(0).ToString().Length)
                    {
                        reader.Close();
                        reader.Dispose();
                        con.Close();
                        return true;
                    }
                    else
                        continue;
                }
                goto skiptoend;
            }
        skiptoend:
            {
                str2 = str2.Remove(str2.Length - 1);
                var check_arsebiti2 = new SqlCommand("SELECT DISTINCT [SubWord]
FROM[GeoLang].[dbo].[Arsebiti] Where SubWord like N'" + str2 + "'", con);
                reader2 = check_arsebiti2.ExecuteReader();
                if (reader.HasRows)
                {
                    while (reader2.Read())
                    {
                        if (compareTwoString(str, reader2.GetValue(0).ToString()) &&
str.Length >= reader2.GetValue(0).ToString().Length)
                        {
                            reader.Close();
                            reader.Dispose();
                            reader2.Close();
                            reader2.Dispose();
                            con.Close();
                            return true;
                        }
                        else
                            continue;
```

```
                }
                return false;
            }
            else
            {
                reader.Close();
                reader.Dispose();
                reader2.Close();
                reader2.Dispose();
                con.Close();
                return false;
            }
        }
    }
```

#14

```
//ფუნქცია რომელიც აბრუნებს არსებითი სახელების მასივს
    public NameValueCollection arsebitiSaxeli (NameValueCollection value)
    {
        NameValueCollection value2 = new NameValueCollection();
        foreach (string s in value)
        {
            foreach (var v in value.GetValues(s))
            {
                if (checkArsebitiSaxeli(s))
                {
                    value2.Add(s, v);
                }
                else
                    continue;
            }
        }
        return value2;
    }
```