VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Automatizuotas gramatinių struktūrų mažinimas lietuvių kalboje

Atliko:
Indrė Jaloveckienė parašas

Vadovas:
dr. Linas Bukauskas

Vilnius
2018

Turinys

Santrauka

Šiame darbe pristatomas metodas lietuviško teksto santraukos gavimui bei metodo įgyvendinimo žingsniai. Pagrindinis darbo tikslas yra sukurti metodą, kuris sugebėtų ne tik ištraukti iš teksto esminius sakinius, bet taip pat sugebėtų tuos sakinius perrašyti. Tokiu būdu tikimės gauti itin tikslią ir trumpą santrauką - tokią, kuri būtų kuo panašesnė į parašytą žmogaus. Šiame darbe, pirmiausia, nagrinėjame kitų autorių nuveiktus darbus, pabrėždami jų silpnąsias ir stipriąsias vietas. Toliau, apibendrintai išdėstome savo teorinį metodą, lietuviško teksto suvedimui į santrauką. Galiausiai, aprašome savo metodo įgyvendinimą, pasiremdami teoriniu modeliu ir nurodydami technologijas, kurias naudojome.

Summary

Automated text summarization for Lithuanian language

In the following work we present a method for automated text summarization for Lithuanian language. The main goal here is to create a method and later on a program for automating text's summarization. As opposed to most currently existing summarization methods, we aim to create a method that would generate an abstractive summary rather than extractive. As input for the program we will have a text in lithuanian and as an output we expect to get a very short but precise text, that would be as close to the human written summary as possible.

Firstly, we did a research on existing summarization methods and pointed out the weakest and the strongest points of each. Then we created a theoretical model for generating lithuanian summary from a text. What is new in our method is that we included syntactic analysis of sentences, not only statistical analysis. With this addition we expected to get a summary that is much more precise and closer to human written text.

In further sections, we have described the implementation of our model. We did text preprocessing in *Python* programming language as it already has quite a lot of built in Natural language processing tools. Later on all data is imported to a *Microsoft SQL Server* database and the rest of analysis is done there.

To sum up the work that has been done, we created a model for lithuanian summary generation from a text. In addition, we started to implement the method programatically but came to an issue that syntactic analysis isn't fully done for lithuanian language and because of that we were not able to fully implement and test our model. However, we presented the means how model could be tested and evaluated in the future. In the meantime, the author of this work suggests to think carefully before choosing abstractive summary methods over extractive - the first one might be more effective but the latter is the one that can give quick results with the least efforts.

Įvadas

Kaip [1] ir [2] šaltiniai teigia, natūralios kalbos apdirbimas (NLP - Natural Language Processing (angl.)) - tai dirbtinio intelekto metodas, leidžiantis mums bendrauti su išmaniosiomis sistemomis, naudojant savo įprastą kalbą, tokią, kaip anglų ar lietuvių. Tokio metodo įvestis ir išvestis gali būti rašytinis arba įgarsintas tekstas.

Domėjimasis kalbos apdirbimu prasidėjo jau daugiau nei prieš pusšimtį metų, tačiau tai vis dar gana neatrastas laukas, keliantis daug klausimų ir sunkumų. Pagrindinė tokių sunkumų priežastis yra natūralios kalbos sistemiškumo trūkumas - egzistuoja mažai taisyklių, tiksliai apibrėžiančių kalbą. Kaip priešpriešą natūraliai kalbai, galime paimti bet kurią iš programavimo kalbų - jos buvo specialiai sukurtos žmogaus, atlikti tam tikras funkcijas. Kiekvienas kodo simbolis, kiekviena eilutė yra lengvai interpretuojama, kadangi buvo parašyta pagal griežtas tos kalbos taisykles. Grįžtant prie natūralių kalbų, jos vis dar vystosi ir nuolatos kinta, taip pat, egzistuoja begalė taisyklių, kaip sudaromi žodžiai ar sakiniai, yra daug išimčių. Negana to, kiekviena kalba turi skirtingas taisykles, skirtingus simbolius, todėl turi būti tiriama ir apdorojama atskirai.

Šiuolaikinėse sistemose NLP yra labai dažnai sutinkamas ir plačiai naudojamas. Sugebėti perskaityti ir suprasti kalbą yra būtina, norint efektyviai ir tiksliai grąžinti paieškos rezultatus, teisingai išversti dviprasmiškus sakinius bei norint, kad mūsų užprogramuoti robotukai ir sistemos elgtųsi pagal mūsų duodamas balsines komandas. Prie kitų pritaikymo sričių galima paminėti ir automatinį teksto apibendrinimą, automatinį tekstų kūrimą bei roboto kurto teksto atpažinimą, automatinį teksto klaidų taisymą, teksto įgarsinimą bei atvirkštinę funkciją - girdimo teksto užrašymą. Iš realių tokių pritaikymų pavyzdžių galima paminėti Apple dirbtinį intelektą "Siri" ar Windows "Cortana", Google vertėją ir vertėjo įgarsinimo funkciją. Naršant internete galima rasti ir dar daugiau įvairių išmaniųjų sistemų, dirbančių su natūraliomis žmonių kalbomis.

Šiame darbe nagrinėsime vieną iš natūralios kalbos apdirbimo tikslų - teksto supaprastinimą arba samarizaciją (nuo angliško žodžio "summary" - santrauka) lietuvių kalbai. Įvairių šaltinių ir teksto samarizacijos implementacijų galima rasti internete daugeliui kalbų, tačiau tai nėra padaryta lietuvių kalbai (kiek žinoma šio darbo autorei) dėl to tai yra aktualu ir reikalinga. Mūsų tikslas yra iš turimo lietuviško teksto gauti kitą tekstą, kuris būtų kiek įmanoma trumpesnis, tačiau išlaikytų tą patį turinį (prasmę) ir būtų lengvai skaitomas bei suprantamas. Be to, visi sudėtingi, ilgi, sudėtiniai sakiniai turėtų būti išskaidyti į mažesnius ir gerokai paprastesnius. Kadangi lietuvių kalboje apstu įvairių metaforų ir perkeltinių prasmių gali būti naudinga jas pakeisti paprastesniais išsireiškimais. Viso darbo siekiamybė yra gauti tekstą, kuris būtų labai trumpas, suteiktų tiek pat informacijos, kiek ir pradinis ir taip pat išliktų rišlus bei skaitomas, t.y., gauta santrauka turėtų būti panaši į parašytą žmogaus.

Egzistuoja trys pagrindiniai automatinių samarizacijų tipai. Pirmasis tipas yra teksto ištraukomis paremta santrauka - tai tokia santrauka, kurioje visi sakiniai ir frazės yra išimti tiesiogiai iš teksto ir nėra niekaip keičiami ar perdaromi. Šios santraukos gali būti skirtos išgauti pagrindines frazes ir raktinius žodžius tekste, arba sutrumpinti turimą tekstą, paliekant tik esminius sakinius. Antrasis samarizacijų tipas yra paremtas apibendrinimais - tai toks būdas, kai sakiniai nėra tiesiogiai kopijuojami iš turimo teksto, bet yra perfrazuojami. Šio tipo santraukos paprastai būna gerokai trumpesnės, bet taip pat gerokai išraiškingesnės ir labiau panašios į parašytas žmogaus. Be to, apibendrinimais paremtos samarizacijos reikalauja natūralios kalbos suvokimo ir sintaksinės analizės, dėl ko kurti tokius metodus yra kur kas sudėtingiau. Paskutinis tipas yra pusiau automatinės samarizacijos. Kaip pavadinimas nurodo, tai nėra pilnai automatizuota santrauka, o tik pagalbinis žingsnis, padedantis vartotojui sudaryti santrauką. Tai gali būti santraukos reikalaujan-

čios žmogaus įvesties prieš sudarant santrauką (pavyzdžiui nurodant kandidatus, kurie turėtų būti įtraukiami) arba santraukos, kurios reikalauja žmogaus įsikišimo po santraukos sudarymo (nurodo vartotojui, kurie sakiniai ar frazės gali būti esminiai).

Savo darbe kursime mišrų metodą, kuriame naudosimės tiek ištraukomis iš teksto, tiek apibendrinimais. Tokiu būdu bandysime sujungti abiejų metodų privalumus bei sumažinti abiejų trūkumus. Pirmiausia sumažinsime sintaksinės analizės apimtis metodo kūrime - atliksime tik minimalią anlizę, kad gautume tik esminius taškus. Taip pat, įvesdami bent minimalią sintaksinę analizę, patikslinsime ištraukomis paremtą samarizaciją. Taip tikimės gauti metodą, kuris būtų santykinai nesudėtingas, bet taip pat - kiek galima tikslus ir išraiškingas.

Pirmiausia, mūsų tikslui pasiekti bus pateikiama susijusių darbų analizė - apžvelgsime, ką kiti autoriai nuveikė šioje srityje bei ką galime iš jų darbų panaudoti savo darbe. Šią analizę galima rasti 1 skyriuje. Toliau, 2 skyriuje, supažindinsime su mūsų darbe naudojamomis sąvokomis ir teorija, pristatysime keletą populiaresnių samarizacijos algoritmų. Be to, po esamų algoritmų apžvelgimo, pateiksime savo teorinę idėją, kaip turėtų veikti mūsų metodas. Sekančiame skyriuje, aprašysime savo modelio kūrimo etapus, supažindinsime su problemomis, su kuriomis teko susidurti bei pateiksime gautus rezultatus. Galiausiai, aprašysime savo išvadas, kylančias iš padaryto eksperimento, bei pateiksime tolimesnes sritis, kurias galima tobulinti, kad šis metodas būtų dar efektyvesnis.

1. Susiję darbai

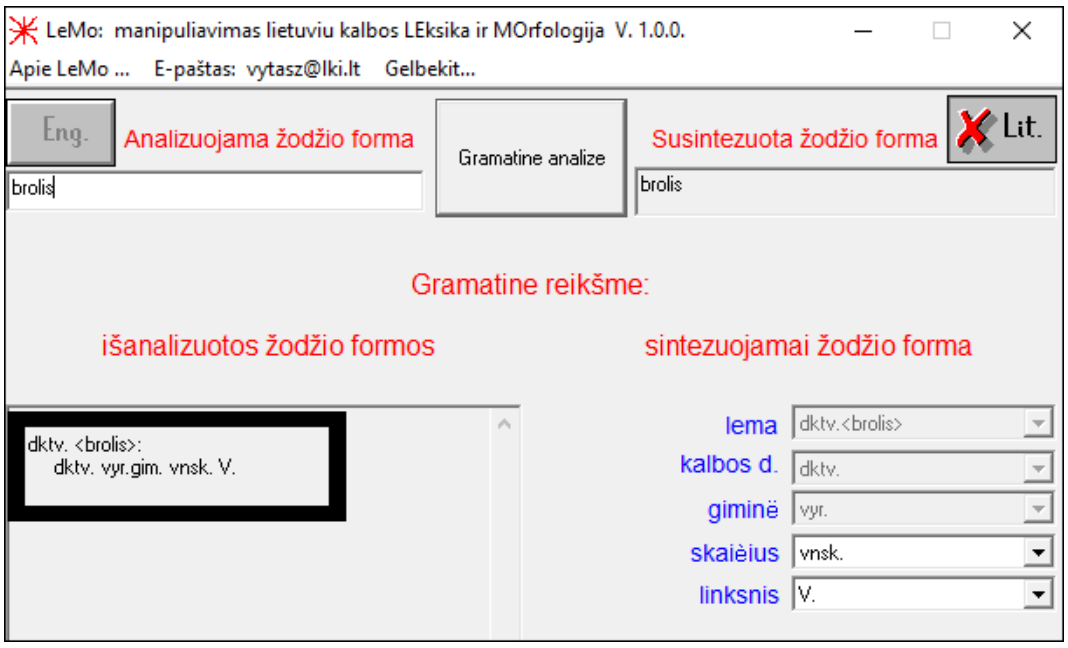
Kiekvieną dieną internetas pasipildo vis naujais straipsniais iš mokslinių sričių bei naujais grožinės literatūros kūriniais ar kitais tekstais. Turint tokį milžinišką naujienų srautą, darosi vis sudėtingiau sekti paskutines naujienas ir suspėti perskaityti visus norimus tekstus. Be to, šiuolaikinė visuomenė darosi vis nekantresnė ir nori gauti paskutines naujienas čia ir dabar. Į pagalbą mums ateina kompiuterinės programos, kurios sugeba turimą ilgą tekstą perskaityti, išanalizuoti ir pateikti ganėtinai trumpą teksto apibendrinimą.

Teksto trumpinimas arba apibendrinimas, kaip ir bet kuris kitas kalbos apdirbimo tikslas, yra itin aktualus lietuvių kalboje, kadangi šioje srityje vis dar trūksta teorijos ir nuveiktų darbų. Nėra sudėtinga rasti įvairių metodų bei implementacijų anglų kalbos apdorojimui. Vien Python programavimo kalba turi milžinišką kalbos apdorojimo biblioteką (*NLTK*), kurios dėka galima rasti angliškų tekstų testavimui, funkcijų sakinių grafams sudaryti ir kitų naudingų metodų. Apie šią biblioteką plačiau galima pasiskaityti internetinėje knygoje *"Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit"* [10]. Be to, daug naudingų pavyzdžių ir metodų kitoms kalboms galima rasti GitHub platformoje.

Viena iš priežasčių dėl kurių trūksta metodų lietuvių kalbos apdorojimui, yra lietuvių kalbos formų gausa - mūsų veiksmažodžių forma priklauso ir nuo laiko, ir nuo giminės, ir nuo nuosakos ("keliauti", "keliaučiau", "keliaus", "keliavome" ir t.t.). Be to, lietuvių kalboje nėra svarbi žodžių tvarka, tai yra, mūsų veiksnys gali būti, tiek pradžioje sakinio, tiek gale, tiek viduryje. Ta pati situacija yra ir su kitomis sakinio dalimis. Pavyzdžiui sakinys "Po ilgos darbo dienos Petras ėjo namo" yra tiek pat taisyklingas ir naudojamas kaip ir sakinys "Petras ėjo namo po ilgos darbo dienos". Taigi, yra itin sudėtinga apibendrintai ir stuktūrizuotai išrašyti visas sakinių sandaros taisykles, kuriomis galėtume pasikliauti. Tiesa, Daivai Šveikauskienei pavyko suformuluoti didžiąją daugumą tokių taisyklių savo daktaro disertacijoje [18]. Savo darbe autorė aprašė paprastųjų vientisinių lietuvių kalbos sakinių sintaksinę analizę ir pasiekė labai gerus rezultatus. Jos metodo tikslumas svyruoja nuo 86 iki 100 procentų, priklausomai nuo pasirinkto teksto pobūdžio (respublikinė periodika, mokslinė periodika, grožinė literatūra, filosofijos tekstai ir panašiai), su vidutiniu 92 procentų tikslumu. Šis metodas labiausiai pasiteisno grožinės literatūros tekstams. Toliau savo darbe ne kartą prisiminsime šią autorę bei naudosimės jos įžvalgomis ir sudaryta lietuvių gramatika.

Šalia D. Šveikauskienės verta paminėti ir Vytauto Zinkevičiaus lemuoklį, aprašytą [19] šaltinyje. Tai yra išsamiausias ir pilniausias lietuvių kalbos lemuoklis. Jo pagalba lietuviški žodžiai tekstuose yra atstatomi į jų pradinę būseną arba lemą. Be to, papildomai gali būti pateikiama ir gramatinė informacija apie žodį - jo kalbos dalis, vienaskaita ar daugiskaita, giminė. Pradinė lemuoklio versija aprašyta [19] šaltinyje apdorodavo tekstinį failą ir rezultatą taip pat išvesdavo į kitą tekstinį failą, kas nėra patogu galutiniam vartotojui. Šiuo metu galima rasti patogesnę lemuoklio versiją *exe* failo pavidalu, kurį galima parsisiųsti iš oficialaus V. Zinkevičiaus internetinio puslapio - *http://donelaitis.vdu.lt/~vytas/lemo/liet/lemo_pasiimt.htm*. Atnaujintas lemuoklis jau priima tiesiog vieną žodį ir visą informaciją apie jį išveda į programos dialogo langą (žiūrėti paveiksliuką 1).

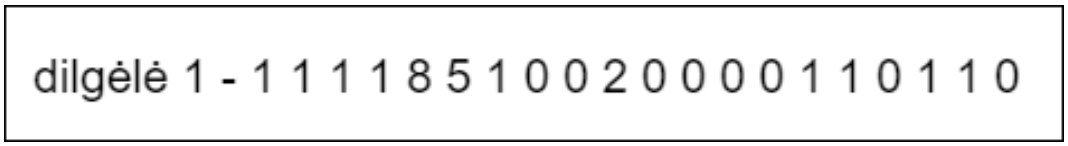
Pagrindinė bet kurio lemuoklio problema yra žodžių dviprasmiškumas. Tai reiškia, kad yra daug žodžių, kurie turi skirtingas reikšmes, tačiau gramatiškai užrašomi taip pat. Tokių žodžių pavyzdys būtų - "*kovas*" (reikšmė gali būti arba mėnesio pavadinimas, arba daugiskaita žodžio "*kova*") arba "*mes*" (naudojamas kaip įvardis arba kaip veiksmažodis). Ši problema yra minima [12] šaltinyje. Čia pristatomas HMM (Hidden Markov Model) modelis pritaikytas lietuvių kalbai



1 pav. V. Zinkevičiaus programos naudojimo pavyzdys.

su tam tikrais pakeitimais. Jo pagalba, turint 1 milijono žodžių tekstyną, yra anotuojami tekstai, kurių nėra tame tekstyne. Kiekvienam žodžiui yra tikimybiškai priskiriama jo lema. Šis modelis pasiekia 94% tikslumą atspėjant žodžio kalbos dalį ir 99% tikslumą spėjant žodžio lemą. Šio modelio pagalba galima tikimybiškai įvertinti žodžio lemos atspėjimą dviprasmiškuose atvejuose ir pasirinkti tą variantą, kuris yra labiau tikėtinas.

Minėtasis V. Zinkevičiaus lemuoklis nėra vienintelis sukurtas lietuvių kalbai. Kita alternatyva šiam lemuokliui yra atviro kodo programa Python programavimo kalbai, kurią galima rasti [9] šaltinyje. Kaip ir pirmojo lemuoklio atveju, čia taip pat gauname visą reikalingą informaciją - žodžio lemą, giminę, skaičių ir kitas charakteristikas. Šios programos privalumas yra kalbos dalių aprašymo lankstumas - kiekviena kalbos dalis yra aprašyta tik tai daliai būdingomis charakteristikomis, tačiau visos kalbos dalys vieningai sudėtos į gramatinio medžio struktūrą. Tokiu būdu, yra nesudėtinga keliauti per medžio šakas, ir susirinkti visas reikalingas charakteristikas. Be to, turime nemažą leksemų sąrašą su joms priskirtomis savybėmis, kurios priskirtos remiantis medžio struktūra. 2 paveiksliuke pateikiame vieną iš tokių leksemų pavyzdžių. Nesudėtinga interpretuoti turimą skaičių seką - pirmas skaičiukas visada nusako kalbos dalį (1 - daiktavardis, 2 - būdvardis ir t.t.), už brūkšnio esančių skaičių reikšmė priklauso nuo prieš tai buvusių savybių, tačiau jas galime atsirinkti keliaudami per minėtą gramatinį medį. Mūsų atveju, sekantys nariai pasako, kad tai yra tikrinis daiktavardis, negalintis turėti neigiamo priešdėlio, moteriškos giminės ir t.t. Savo darbe dalinai remsimės tokia medžio struktūra, tik gerokai ją supaprastinsime, kadangi mums nėra aktualu kiekviena smulki kalbos dalies charatkteristika, o tik apibendintas aprašymas.



2 pav. Python programos leksemos pavyzdys.

Dar vieną lietuviškų žodžių formų žodyną galima rasti internete, adresu *www.morfologija.lt*. Kaip ir kituose lemuokliuose, neišvengiame problemos, kad gautos žodžio kalbos dalys nėra vie-

nareikšmiškai nusakomos (šią problemą plačiau nagrinėsime 3.3 skyrelyje). Čia taip pat galima rasti žodžio lemą bei kitas to žodžio gramatines formas. Šios platformos pliusas yra tas, kad jinai pasiekiama internetu ir nereikia diegti papildomų programų, o taip pat, ją galima tobulinti patiems, pridedant naujus žodžius ar žodžių formas. Pagrindinis platformos minusas - šis žodynas nėra ypatingai platus, todėl atsiranda nemažai žodžių, kurių negalime rasti. Be to, tenka dirbti su nemažu HTML kodu ir iš jo atsirinkti tik tas dalis, kurios mums reikalingos, praleidžiant teksto formatavimą ir panašiai. Visgi, savo darbe toliau naudosimės šia platforma, kadangi tai yra greičiausias ir paprasčiausias būdas gauti žodžių gramatines dalis, nepaisant kitų minusų.

Vienas pirmųjų metodų ir eksperimentų, skirtas automatinei teksto santraukai, buvo pristatytas dar 1958 metų kovą Nacionaliniame suvažiavime, Niujorke. Vėliau, eksperimento ir metodo detalės buvo publikuotos IBM žurnale - straipsnyje "The Automatic Creation of Literature Abstracts" ("Automatinis literatūros santraukos kūrimas") [16]. Šio straipsnio autorius - H. P. Luhn - nagrinėjo grynai statistinį metodą, kuris leistų sugeneruoti straipsnio santrauką, remiantis vien tik žodžių dažnumu tekste bei dažniausiai pasikartojančių žodžių išsidėstymu sakinyje. Tokiu būdu, autorius norėjo palengvinti straipsnių kūrėjams jų darbą, sumažindamas reikalingų žmogiškųjų išteklių kiekį, rašant straipsnio santrauką. H. P. Luhn išdėstytame metode, visi bendrašakniai žodžiai yra laikomi sinonimais - nėra kreipiamas dėmesys žodžio formoms ir kitoms išraiškoms. Be to, joks dėmesys nėra skiriamas logikai tekste ar semantinių ryšių nagrinėjimui - jokia žodžių ar frazių tekste prasmė nėra įtraukiama į metodą. Šis metodas yra pilnai ištraukomis paremto metodo pavyzdys.

Dar vienas iš ištraukomis paremtų samarizacijų metodų yra *TextRank* algoritmas, tumpai ir gana aiškiai išdėstytas [3] šaltinyje. Šis algoritmas paremtas kitu gerai žinomu algoritmu *PageRank*, kuris naudojamas Google paieškos sistemoje. *PageRank* (nors ir gerokai perdarytas) naudojamas apskaičiuoti internetinio puslapio įvertį, pagal kurį yra pateikiami ir išrikiuojami paieškos rezultatai. Metodas vadovaujasi pagrindine idėja, kad svarbūs puslapiai turi nuorodas į kitus svarbius puslapius (kuo daugiau svarbių nuorodų puslapis turi, tuo pats puslapis svarbesnis), o puslapio įvertis iš tikrųjų yra vartotojo tikimybė apsilankyti tame puslapyje. Pritaikydami *PageRank* algoritmą santraukos ieškojimui, vietoj puslapių nuorodų imame teksto sakinius. *TextRank* algoritmas nepriklauso nuo pasirinktos kalbos ir nereikalauja jokio apmokymo dėl to yra patogus naudoti.

Kaip jau minėjome pradžioje, yra keletas skirtingų būdų santraukai sugeneruoti - arba išrinkti tik išraiškingiausias, informatyviausias teksto dalis (kaip frazės, žodžiai ar sakiniai), arba perfrazuoti turimus sakinius naujai. Mūsų atveju reikia kiek nestandartinio požiūrio į šią problemą. Labai įdomus samarizacijos metodas aprašytas [13] šaltinyje. Čia autoriai nagrinėja prekių atsiliepimų ir komentarų apibendrinimą, vaizdžiai pavadintą *micropinion* (micro-nuomone). Kertinis akmuo šiame metode yra n-gramų naudojimas naujo teksto generavimui, pasiremiant tų n-gramų dažnumu. Čia pirmiausia yra išrenkamos dažniausios unigramos visuose komentaruose ir naudojant šias unigramas yra auginamos aukštesnio laipsnio n-gramos iki vartotojo pasirinkto lygio. Pagrindinis ir labai svarbus šio metodo pliusas yra tas, kad jis visiškai nereikalauja sintaksinės analizės, tačiau jam reikalingas n-gramų žodynas.

Aukščiau paminėti samarizacijų metodai yra tik keletas pavyzdžių iš gana nemažo jų sąrašo. Tiesa, kai kurios implementacijos veikia geriau negu kitos, tačiau visais atvejais atsiranda tam tikrų metodo trūkumų - dėl to geriausio metodo paieškos tebesitęsia. Nemaža dalis tyrėjų bando lyginti tarpusavyje jau egzistuojančius metodus, bei bando juos jungti, tikėdamiesi gauti geresnius rezultatus. Gana neseniai (2017 metų gegužę) viena populiariausių ryšių su klientais valdymo kompanija - *SalesForce* - išleido pranešimą [4], jog jiems pavyko sukurti itin efektyvų metodą ilgų tekstų santraukos gavimui. Jų metodas paremtas neuroniniais tinklais ir sugeba sugeneruoti sant-

raukas ne tik išskirdamas originalius sakinius iš teksto, bet taip pat geba sakinius perfrazuoti. Šis metodas naudoja du mašinų mokymo metodus - *Reinforcement learning* (sustiprinimo) ir *Teacher Forced learning* (mokytojo). Tokiu būdu kūrėjai sugebėjo padidinti įprastų metodų jautrumą bei galutinio teksto nuoseklumą ir rišlumą. Tiesa, tikslesnis algoritmo aprašymas kol kas nėra viešai prieinamas (kiek žinoma šio darbo autorei.)

Gana plačią įvairių metodų apžvalgą autoriai pateikia [11] straipsnyje. Čia aprašoma keletas santraukos radimo metodų, tarp jų ir neuroninių tinklų naudojimas. Tiesa, čia nerasime atsakymo, kuris metodas yra pats efektyviausias, tačiau bent jau galime susipažinti su galimais problemos sprendimo būdais. Paprastai, apibendrinimais paremtos santraukos būna gerokai glaustesnės ir išraiškingesnės, tačiau reikalauja sudėtingos implementacijos, o tokie metodai yra mažai pritaikomi platesnei aibei tekstų. Tuo tarpu ištraukomis paremtos santraukos yra greitai įgyvendinamos ir duoda pakankamai neblogus rezultatus. Dėl šios priežasties, šis metodų tipas yra populiariausias tiek tarp tyrėjų, tiek tarp kompanijų, naudojančių automatines santraukas.

2. Teorinis modelis

Šiame skyriuje supažindinsime su pagrindiniais natūralios kalbos apdorojimo terminais ir sąvokomis. Taip pat pristatysime jau egzistuojančius tyrimo metodus, kuriuos naudosime savo darbe. Susipažinę su jau egzistuojančiais metodais, aprašysime savo metodą, lietuviško teksto santraukos radimui.

2.1. Nuo konteksto nepriklausanti gramatika

Skaitydamas tekstą žmogus automatiškai analizuoja, kas yra parašyta - intuityviai nustato sakinio veikėją bei jo daromą veiklą, visas susijusias aplinkybes. Kaip daugelis prisimename iš mokyklos, bet kokia sakinio analizė prasideda nuo sakinio veiksnio (kas?) ir tarinio (ką veikia?) nustatymo, todėl tokią analizę reikia įtraukti ir į automatizuotą teksto trumpinimą, tačiau tai nėra taip paprasta ir intuityvu kai bandome tai aprašyti programatiškai. Kompiuteris negali intuityviai kaip žmogus suvokti, apie ką yra tekstas, kas yra teksto veikėjas ir ką jis veikia. Kad tą sugebėtų, jam reikia tam tikrų taisyklių, kurios apibrėžtų galimus kandidatus į sakinio veikėjus ir kitas sakinio dalis. Čia į pagalbą ateina gramatikos ir sakinių grafai, kurie leidžia mums analizuoti turimą tekstą, nesuprantant paties teksto turinio - dėl to tai ir vadinama nuo konteksto nepriklausančia gramatika.

```
1. S -> NP VP
2. VP -> V NP
3. NP -> N | ART N
```

3 pav. Anglų kalbos gramatikos pavyzdys.

Žodį *"gramatika"* paprastai įsivaizduojame, kaip kalbotyros šaką ir taisyklių rinkinį, pagal kurį nusprendžiame, kaip taisyklingai rašyti. Natūralios kalbos apdorojime šis terminas įgyja kiek kitokią prasmę - tai vis dar taisyklių rinkinys, tačiau jis yra gerokai supaprastintas ir sudarytas taip, kad būtų lengvai pritaikomas kompiuterinėms programoms. Be to, tokios gramatikos gali būti sudarinėjamos, ne tik įprastinėms žmonių kalboms (3 pav.), bet ir programavimo kalboms (4 pav.). 3 paveiksliuke turime paprasto, trumpo angliško sakinio gramatiką. Pirmoji eilutė nusako, kad sakinys (**S**) turi būti sudarytas iš daiktavardinės frazės (**NP** - noun phrase (angl.)) ir po jos einančios veiksmažodinės frazės (**VP** - verbal phrase (angl.)). Toliau, **VP** gali būti sudaryta tik iš veiksmažodio ir kitos daiktavardinės frazės, o **NP** gali būti sudarytas arba iš daiktavardžio, arba iš artikelio su daiktavardžiu. Analogiškai, 4 paveiksliuke aprašoma progamavimo kalbos *for* ciklas. Pirmoji eilutė sako, kad **ForStatement** (ciklas), turi būti sudarytas iš paryškintų raktinių žodžių, einančių drauge su ciklo pradžia ir pabaiga (*start* ir *end*) ir kokiu nors **StatementBlock** (sakinių bloku). Atitinkamai, **StatementBlock** gali būti pakeičiamas taip pat į **ForStatement** arba tiesiog į **Statement**. Atkreipiame dėmesį, kad šiuo atveju antra ir trečia eilutės galėtų būti sudėtos į vieną ir atskirtos vertikaliu brūkšniu, kaip prieš tai nagrinėto pavyzdžio trečioje eilutėje - tai yra du lygiaverčiai žymėjimai.

```
1. ForStatement -> for start to end begin StatementBlock end
2. StatementBlock -> ForStatement
3. StatementBlock -> Statement
```

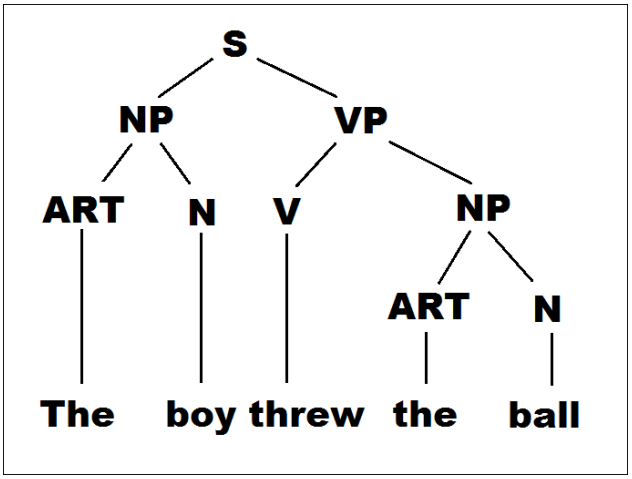
4 pav. Programavimo kalbos gramatikos pavyzdys.

1 apibrėžimas. Backus - Naur forma (arba Backus normine forma) vadiname rinkinį taisyklių, turinčių tokią formą:

$$< symbol > ::= \_\_expression\_\_, \tag{2.1}$$

kur "$\_\_expression\_\_$" žymi simbolių seką, o du dvitaškiai ir lygybė (" ::= ") žymi pakeitimo/praplėtimo operaciją. Kairėje esantis simbolis gali būti pakeistas arba praplėstas išraiška, esančia dešinėje pusėje. Be to, dešinėje pusėje gali būti ir daugiau negu viena simbolių seka - tokiu atveju, tas sekas atskiriame vertikaliuoju brūkšniu ("|"). Jeigu dešinėje pusėje yra daugiau negu viena simbolių seka, vadinasi, kairė pusė gali būti pakeista bet kuria simbolių seka iš dešinės pusės. Trumpumo dėlei šią formą vadinsime tiesiog BNF.

Paprastai tariant, BNF yra gramatikos žymėjimo arba užrašymo metodas. Taigi, prieš tai mūsų sutikti du gramatikų pavyzdžiai atitinka BNF formą ir juos galima perrašyti analogiškai pagal 1 apibrėžimą. Kitas gramatikos aprašymo būdas yra sakinių medžiai (*"Parse Tree"* angl.) arba grafai - struktūra pavaizduota 5 paveiksliuke. Pačiame giliausiame (žemiausiame) medžio lygyje atsiranda sakinio žodžiai, o einant medžiu į viršų, sutinkame sakinio dalis (*V* - veiksmažodis, *N* - daiktavardis) ir kitas leksines detales (*ART* - artikelis). Pačiame viršuje *S* raide pažymėta pradžios simbolis. Lygindami šį medį su 3 paveiksliuke pavaizduota gramatika, matome, kad medis atitinka gramatiką - S pakeičiama į NP ir VP, NP pakeičiama į ART N, o VP pakeičiama į V ir NP. Tik šiuo atveju, atsiranda dar vienas sluoksnis - patys sakinio žodžiai. Metodai ir algoritmai, skirti šiems medžiams sudaryti, yra vadinami analizatoriais arba parseriais ir gali būti naudojami sintaksinei analizei palengvinti. Daugiau pasiskaityti apie gramatikas ir sakinių medžius galima internete, [17] šaltinyje. Čia aiškiai ir glaustai pateikta keletas medžių sudarymo metodų ir taip pat šiek tiek informacijos bei pavyzdžių apie pačias gramatikas.



5 pav. Sakinio medžio pavyzdys.

Sudaryti tokią gramatiką bet kuriai žmonių kalbai yra labai sudėtinga. Tai reikalauja daug kalbos gramatikos žinių ir įgūdžių bei daug žmogiškųjų išteklių darbo. Taip yra todėl, kad sudaryta gramatika turi atspindėti visus įmanomus sakinius toje kalboje, kurių gali būti be galo daug. Tiesa, vienoms kalboms gramatika gali būti sudaroma lengviau negu kitoms. Pavyzdžiui, anglų kalboje veiksnys turi savo fiksuotą vietą ir visuomet eina pradžioje sakinio, dėl to dalis sakinių variantų automatiškai atkrenta, kadangi neįmanomi tokie sakiniai, kur tarinys eitų prieš veiksnį. Tuo tarpu lietuvių kalboje veiksnys gali būti bet kurioje pozicijoje ir aprašydami gramatiką turime išnagrinėti visus variantus, kur tarinys eina po veiksnio ir kur veiksnys eina po tarinio. Taip pat, turime

nepamiršti ir kitų sakinio dalių, kurios irgi gali atsirasti bet kurioje sakinio vietoje. Tokią gramatiką lietuvių kalbai jau sudarė D. Šveikauskienė bei aprašė savo daktaro disertacijoje [18]. Tiesa, ši gramatika aprašo tik paprastuosius vientisinius sakinius, dėl to, pilnai neatspindi visos lietuvių kalbos. Norint naudoti šią gramatiką savo darbe, turime ją papildomai pritaikyti - arba skaidyti sudurtinius sakinius į vientisinius, arba praplėsti šią gramatiką likusiais sakinių variantais.

2.2. N-gramos

Žodis *"gram"* gali būti kildinamas iš graikų kalbos žodžio, reiškiančio raidę, taigi, paraidžiui *n-grama* nusako *n* raidžių kombinaciją. Internetiniai anglų kalbos žodynai ([5], [6]) šį terminą apibūdina kaip *n* kaimyninių žodžių seką iš duoto teksto arba kaip seką, sudarytą iš *n* simbolių, sudarančių žodį arba žodžių sekas.

2 apibrėžimas. Kompiuterinės lingvistikos srityje, *n-grama* nusako iš eilės einančių komponentų posekį $w_1, w_2, ..., w_n$, su ilgiu n, iš kokios nors sekos. Komponentais gali būti balsės, priebalsės, raidės, žodžiai ir panašiai. Kai $n = 1$, tokią n-gramą vadinsime unigrama, kai *n* = 2 - digrama, o kai *n* = 3 - trigrama. Aukštesnio laipsnio *n-gramos* atskirai savo pavadinimų neturi ir jas tiesiog vadinsime 4-grama, 5-grama ir taip toliau.

Kitaip tariant, *n-grama* nusako žodžių arba simbolių, einančių vienas po kito, sekas, atsirandančias tam tikrame tekste. Šios sekos gali būti frazės, turinčios tam tikrą reikšmę - "knygų autorius", "pagaliau ryžosi", arba gali būti tiesiog žodžiai, atsidūrę vienas šalia kito ir drauge nesudarantys prasmės - "tai kas", "kas jam". Imkime lietuviško sakinio pavyzdį ir sudarykime unigramas, digramas ir trigramas - "Tyli kiaulė - gilią šaknį knisa". Unigramos bus - "tyli", "kiaulė", "gilią", "šaknį", "knisa", digramos - "tyli kiaulė", "kiaulė gilią", "gilią šaknį", "šaknį knisa", trigramos - "tyli kiaulė gilią", "kiaulė gilią šaknį" ir "gilią šaknį knisa". Žemiau, 1 kodo ištraukoje, pateikiame algoritmą, n-gramų generavimui. Čia N žymi sakinio žodžių skaičių, nGramList - galutinį n-gramų sąrašą, o wordList - sakinio žodžių masyvą.

```
1 for int i = 1; i <= N - n +1; i++:
2         curGram = [n];
3         for int j = 0; j < n; k++:
4                 curGram.Add(wordList[i + j]);
5         nGramList.Add(curGram);
```

1 išeities kodas. Algoritmas n-gramų generavimui

Išskirdami n-gramas iš teksto, vėliau galime apskaičiuoti tikimybes vienam žodžiui eiti po kito žodžio. Tokiu būdu, galime patikrinti gramatinį sakinių tikslumą. Pavyzdžiui, sakiniai "vakar stipriai lijo" ir "vakar galingai lijo" atrodo abu gramatiškai teisingi, tačiau turbūt dažniau sutinkame pirmąjį posakį negu antrąjį. Tikrindami digramų "stipriai lijo" ir "galingai lijo" dažnumą tekstuose, pastebėtume, jog pirmoji frazė yra gerokai dažniau sutinkama, negu antroji. Dėl to pirmasis sakinys yra labiau vartotinas ir tikėtina taisyklingesnis negu antrasis.

Be gramatikos tikslinimo, n-gramos gali būti įrankis, naudojamas automatiniame vertime nustatyti vertimo taisyklingumą. Tai ypač aktualu, kai vienas kažkurios kalbos žodis gali būti išverstas į keletą kitos kalbos žodžių. Tokiu atveju, tikrintume tokio žodžio ir šalia jo einančio žodžio poros pasikartojimus įvairiuose tekstuose ir tikimybiškai pasirinktume dažniausiai sutinkamą žodžių porą. Kitas n-gramų pritaikymas - teksto užrašymas iš kalbos. Kartais įrašyta kalba gali būti neaiški ir dėl to užrašytas tekstas neatitiks tikrovės arba bus užrašyti neegzistuojantys žodžiai,

tačiau tikrindami n-gramas, galime atstatyti tikruosius žodžius. Taip pat, n-gramos gali būti naudojamos sentimentų analizėje ar nuspėjant toliau einantį žodį (pavyzdžiui, rašant tekstą į paieškos sistemą, mūsų sakiniai yra automatiškai užbaigiami).

Vertinant sutinkamų frazių dažnumą tekste, reikėtų nepamiršti, kad žodžiai, sudarantys frazę, nebūtinai eina vienas šalia kito. Pavyzdžiui, imkime du sakinius - "Už lango lijo lietus" ir "Už lango lijo merkiantis lietus". Mes iškart pastebime, kad abu sakiniai nusako tą patį faktą - "lijo lietus", tačiau, jei skaičiuotume digramos "lijo lietus" pasikartojimus šiuose sakiniuose, gautume tik vieną pasikartojimą, kadangi antro sakinio atveju, tarp žodžių yra įsiterpęs papildomas žodis. Turint omenyje lietuvių kalbos laisvą žodžių tvarką sakinyje, turėtume tokius atvejus įtraukti į n-gramų skaičiavimus. Iš čia išplaukia kitas apibrėžimas.

3 apibrėžimas. *k-skip-n-grama* yra *n-gramos* apibendrinimas ir bus vadinama n ilgio posekis $w_1, w_2, ..., w_n$ iš kokios nors sekos, kur komponentai w_i gali būti nutolę vienas nuo kito daugiausiai per k.

Taigi, *n-grama* yra iš tikrųjų *0-skip-n-grama*. Prisiminkime prieš tai nagrinėtą sakinį ir sudarykime visas *1-skip-2-gramas* - į sąrašą pateks visos digramos iš prieš tai ("tyli kiaulė", "kiaulė gilią", "gilią šaknį", "šaknį knisa"), plius naujos digramos su tarpais - "tyli gilią", "kiaulė šaknį", "gilią knisa". Žemiau, 2 kodo ištraukoje, pateikiame algoritmą, k-skip-n-gramų generavimui. Čia allSkips masyvas yra $(n-1)$ dydžio masyvų masyvas, kuriame yra visos įmanomos kombinacijos tarpams tarp n-gramos žodžių. Pavyzdžiui, 1-skip-2-grama atveju allSkips = [(0), (1)] - tarpas tarp dviejų bigramos žodžių gali būti arba 1 arba 2; 1-skip-3-grama atveju, allSkips = [(0,0),(0,1),(1,0),(1,1)] - tarpas tarp pirmo ir antro žodžio bus arba 0, arba 1, tarpas tarp antro ir trečio žodžio, taip pat, arba 0, arba 1.

```
1 allSkips = [(k+1)^(n-1)]
2
3 for int i = 1; i <= N - n + 1; i++:
4
5         foreach skips in AllSkips:
6                 totalSkips = 0;
7
8                 for int s = 0; s <= len(skips):
9                         curGram.Add(wordList[i + totalSkips])
10                        if s < len(skips) then
11                                totalSkips += skips[s] + 1;
12                        end if
13                end for
14
15                nGramList.Add(curGram);
16        end for
17 end for
```

2 išeities kodas. Algoritmas n-gramų generavimui

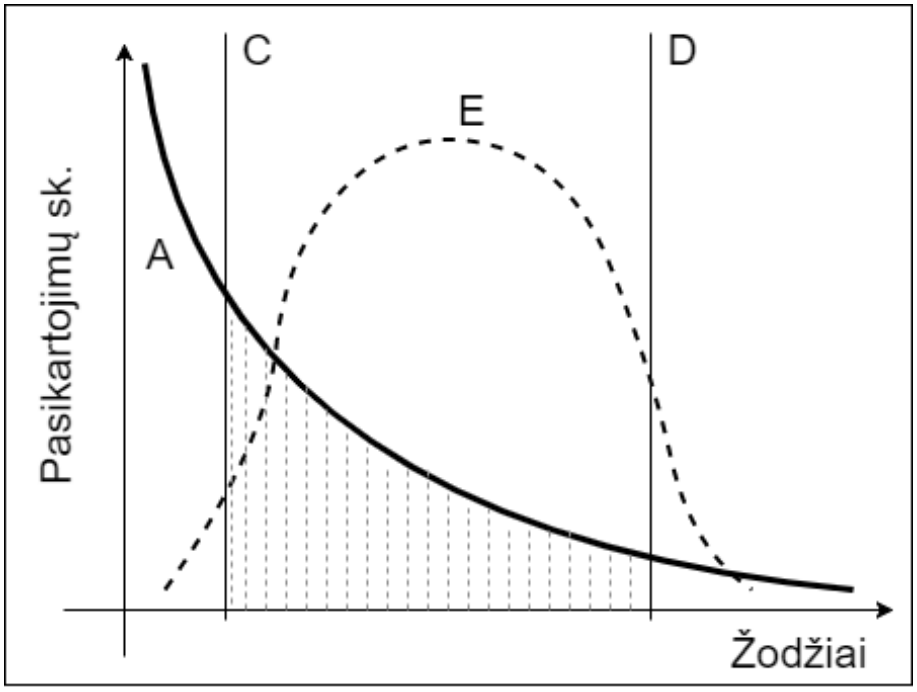
Toliau visame darbe, sakydami digrama ar trigrama (ar *2-grama*, *3-grama*) visuomet turėsime omenyje *K-skip-n-gramas*, kur n bus atitinkamai 2 arba 3, o k bus 3.

2.3. Ištraukomis paremti metodai

2.3.1. H. P. Luhn metodas

Šis metodas yra standartinis, ištraukomis paremtas metodas, ir beje, yra pradininkas tolimesnių metodų. Jo esmė yra surikiuoti sakinius į eilę, pradedant svarbiausiu arba reikšmingiausiu ir baigiant mažiausiai reikšmingu sakiniu. Tuomet, priklausomai nuo implementacijos, pasirinkti tam tikrą skaičių sakinių nuo pradžios, kurie ir sudarys teksto santrauką.

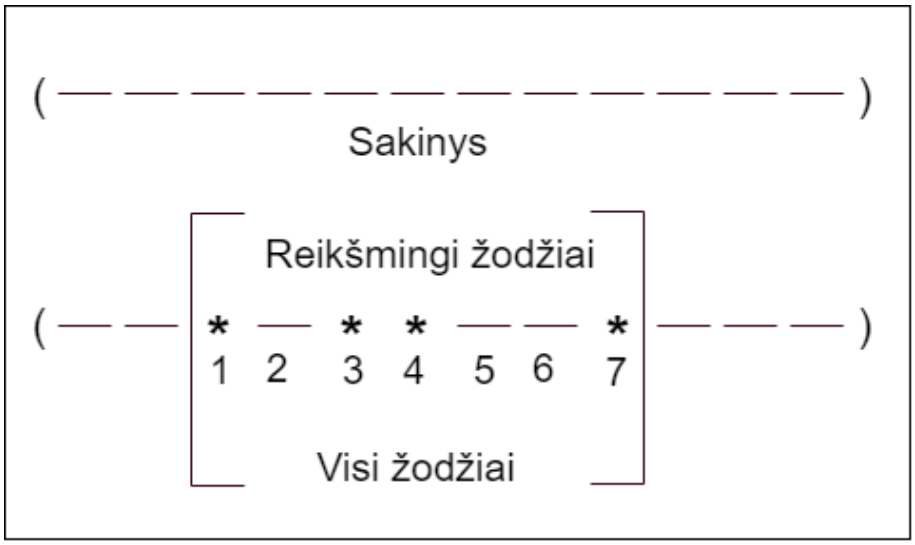
Pirmasis metodo žingsnis suranda reikšmingų žodžių aibę iš duoto teksto. Žodžio reiškmingumą nurodo to žodžio vartojimo tekste dažnumas - kuo dažnesnis žodis, tuo, tikėtina, jis yra reikšmingesnis. Tačiau, reikia nepamiršti žodžių, kurie gali būti sutinkami labai dažnai, tačiau neturi esminės reikšmės tekste (kaip anglų kalboje artikeliai "a" ir "the"). Tokie žodžiai yra vadinami triukšmu ir paprastai jie turi didžiausią dažnį tekste. Šiuos triukšmo žodžius galima atmesti pasinaudojant iš anksto sudarytu žodžių sąrašu, arba, pasiremiant statistine analize, atmesti tam tikrą kiekį dažniausiai pasikartojančių žodžių. Kitaip tariant, užsibrėžiame ribą, iki kurios visi žodžiai pasikartoja per daug kartų, kad būtų svarbūs - šią ribą žymi raidė C, 6 paveiksliuke. Taip pat, reikia užsibrėžti ir viršutinę ribą, už kurios atsirandantys žodžiai pasikartoja per retai, kad būtų svarbūs, dėl to yra nereikšmingi - šią ribą žymi raidė D. Taigi, visi žodžiai 6 paveikslėlyje, papuolantys į tarpą tarp C ir D (užbrūkšniuotas plotas) yra laikomi reikšmingais. Čia A raide pažymėta linija žymi žodžių tekste dažnumą, o punktyrine linija (E) pažymėta žodžių reikšmingumo kreivė - tai hipotetinis svarbių žodžių pasiskirstymas tekste. Ribų C ir D parinkimas gali skirtis priklausomai nuo teksto pobūdžio - kad ribos būtų parinktos teisingai, reikalinga nemaža tekstų analizė.



6 pav. H. P. Luhn metodo, žodžių reikšmingumo grafikas.

Turėdami reikšmingų žodžių aibę, galime įvertinti sakinių reikšmingumo lygį. Sakinių reikšmingumo laipsnio skaičiavimas grindžiamas idėja, kad kalboje užrašytos ar išsakytos panašios mintys dažniausiai atsiduria ir fiziškai šalia. Dėl to manoma, jog du (ar daugiau) reikšmingi žodžiai einantys netoli vienas kito, nurodo to sakinio svarbumą. Tačiau, jeigu tame pačiame sakinyje esantys reikšmingi žodžiai yra per gana didelį atstumą, tai galėtų reikšti, kad šie žodžiai nežymi

svarbos. Pagal daug autoriaus nagrinėtų tekstų analizę, laikoma, kad atstumas tarp dviejų reikšmingų žodžių neturėtų būti didesnis negu 4 ar 5 žodžiai - jeigu tas atstumas yra didesnis, tuomet tie žodžiai nežymi svarbos.



7 pav. H. P. Luhn metodo sakinio įverčio schema.

7 paveikslėlyje pavaizduota sakinio reikšmingumo įverčio skaičiavimo schema. Pirmiausia randami reikšmingi žodžiai sakinyje, kurie nutolę ne didesniu atstumu negu leistinas, ir gaunama sakinio ištrauka, pagal kurią bus skaičiuojamas įvertis (apskliausta sakinio dalis paveikslėlyje). Pažymėkime reikšmingų žodžių apskliaustoje dalyje skaičių raide C, o visų apskliaustų žodžių skaičių - raide T, tuomet sakinio reikšmingumo įvertis (S) skaičiuojamas taip:

$$S = \frac{C^2}{T}; \tag{2.2}$$

Taigi, turėdami kiekvieno sakinio įvertį, galime juos išrikiuoti nuo reikšmingiausio iki mažiausiai reikšmingo. Priklausomai nuo reikalavimų, galime pasirinkti vieną ar keletą svarbiausių sakinių ir juos laikyti teksto santrauka. Jeigu tekstas yra pakankamai didelis, tuomet gali būti naudinga jį padalinti į keletą dalių ir toms dalims ieškoti atskirų santraukų. Tuomet tokio teksto santrauką sudarys keletas mažesnių santraukų. Apibendrindami metodą, galime išskirti pagrindinius pliusus ir minusus:

- Metodas nereikalauja sintaksinės ar loginės teksto analizės;
- Nesudėtinga implementacija;
- Metodas gali būti pritaikomas skirtingo ilgio tekstams;
- Metodas gali būti pritaikomas skirtingo pobūdžio tekstams;
- Reikalinga pradinė analizė, nustatyti parametrams C ir D;
- Reikalinga pradinė analizė, sudaryti triukšmo žodžių sąrašą;
- Metodas kartais gali duoti santrauką, nepilnai atitinkančią tekstą;
- Gauta santrauka gali būti nerišli.

2.3.2. TextRank algoritmas

Prieš gilindamiesi į *TextRank* algoritmą, pirmiausia apžvelgsime jo pirmtaką - *PageRank* - algoritmą. Apsirašykime naudojamus parametrus:

- N - puslapių, su nuorodomis į kitus puslapius, skaičius;
- $P[i]$ - puslapio i įvertis - tikimybė, kad vartotojas užeis į puslapį;
- $C(i)$ - nuorodų, esančių puslapyje i, skaičius;
- $A[i][j]$ - tikimybė, kad vartotojas pereis iš puslapio i į puslapį j, pasinaudodamas nuoroda;
  -
  $$A[i][j] = \frac{1}{C(i)}, \text{jei tarp puslapių } i \text{ ir } j \text{ yra nuoroda;} \tag{2.3}$$
  -
  $$A[i][j] = 0, \text{jei tarp puslapių nėra nuorodos;} \tag{2.4}$$
  -
  $$A[i][j] = \frac{1}{N}, \text{jei puslapis } i \text{ neturi iš jo išeinančių nuorodų;} \tag{2.5}$$

Apsibrėžkime nuorodų tarp puslapių grafą:

$$links = \left\{ \begin{array}{c} page\_1 : set([page\_i]), \\ page\_2 : set([page\_i), \\ ..., \\ page\_N : set([page\_i]) \end{array} \right\} \tag{2.6}$$

Čia $set([page\_i])$ nusako puslapių aibę, į kuriuos yra nuoroda iš vieno iš puslapių kairėje ($page\_1$, $page\_2$, ..., $page\_N$). Ši aibė gali būti ir tuščia. Apsirašome tikimybių matricą, kurios dydis bus $N \times N$:

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \tag{2.7}$$

Dabar, visus a_{ij} apskaičiuojame pagal 2.3 - 2.5 išraiškas. Tuomet galime aprašyti patį *PageRank* algoritmą, kuris turės du parametrus:

- eps - algoritmas sustoja, kai skirtumas tarp dviejų gretimų iteracijų yra mažesnis arba lygus šiam skaičiui;
- d - slopimimo koeficientas. Su tikimybe $d - 1$, vartotojas atsitiktinai pasirenka sekantį puslapį, ignoruodamas bet kokią puslapių nuorodų sistemą;

```
def pagerank(A, eps, d):
    P = ones(N) / N
    while True:
        new_P = ones(N) * (1 - d) / N + d * A.T * P
        delta = abs((new_P - P).sum())
        if delta <= eps:
            return new_P
        P = new_P
```

3 išeities kodas. *PageRank* algoritmo pseudokodas

Čia funkcija $ones(N)$ reiškia $1 \times N$ dydžio matricą, kurios visi elementai lygūs 1, $A.T$ žymi matricos A transpoziciją, o $.sum()$ - visų matricos elementų sumą. Iš 3 kodo ištraukoje pavaizduoto algoritmo gausime $N \times 1$ dydžio matricą, kurios elementai atitinkamai žymės kiekvieno iš puslapių įvertį. Pasinaudodami šiais įverčiais galime išrikiuoti puslapius nuo svarbiausio iki mažiausiai svarbaus.

Toliau *PageRank* algoritmą suvesime į *TextRank* algoritmą, naudodamiesi žingsniais žemiau:

- Puslapių nuorodas pakeičiame sakiniais;
  - Šiuo atveju sakinių grafas bus simetrinis - visi sakiniai, turės "nuorodas" į visus sakinius;
- Matrica A užpildoma analogiškai, skaičiuojant tikimybes iš sakinio i pereiti į sakinį j;
  - Šiuo atveju, tikimybė bus lygi tų sakinių panašumo funkcijos rezultatui;
  - Sakinių funkcijos parinkimas priklauso nuo vartotojo ir gali turėti daug įvairių implementacijų;
- Pritaikome *PageRank* algoritmą, aprašytą 3 kodo ištraukoje, sakinių grafui;
- Iš 3 algoritmo gauname kiekvieno sakinio įvertį. Išrikiavę sakinius pagal įvertį nuo didžiausio iki mažiausio, galime pasirinkti tam tikrą skaičių sakinių nuo pradžios, kurie ir sudarys mūsų santrauką.

Žemiau, apibendrindami metodą, išvardinsime pliusus ir minusus:

- Algoritmas nereikalauja apmokymo;
- Nesudėtinga implementacija;
- Priklausomai nuo pasirinktos panašumo funkcijos, metodas arba priklauso arba nepriklauso nuo pasirinktos kalbos;
- Didelę įtaką daro parametrų eps ir d parinkimas;
- Algoritmo našumas yra stipriai priklausomas nuo pasirinktos panašumo funkcijos.

2.4. Apibendrinimais paremti metodai

2.4.1. Mikro-nuomonių generavimas

Mikro-nuomonės (*micropinion*) generavimas iš tikrųjų yra tarpinis variantas tarp ištraukomis ir apibendrinimais paremtų metodų. Tokiu būdu sugeneruotas tekstas turės tik tuos žodžius, kurie buvo panaudoti originaliame tekste, tačiau frazės ir sakiniai nebus tiesiogiai paimti iš originalo. Dėl šios priežasties, šį algoritmą priskiriame prie apibendrinimais paremtų metodų.

Pirmiausia apsibrėšime pagrindinius žymėjimus:

- $Z = \{z_i\}_{i=1}^{n}$ - sakinių aibė, paduodama į algoritmą;
- $M = \{m_1, m_2, ..., m_k\}$ - grąžinama mikronuomonių aibė, kur kiekvienas m_k yra n-grama;
- $S_{rep}(m_i)$ - įverčio funkcija, matuojanti m_i reprezentatyvumą;
- $S_{read}(m_i)$ - įverčio funkcija, matuojanti m_i skaitomumą;
- σ_{ss} - vartotojo nustatomas parametras, žymintis santraukos ilgį (kuo mažesnė reikšmė, tuo trumpesnė snatrauka);
- σ_{sim} - vartotojo nustatomas parametras, žymintis santraukos nuomonių dubliavimąsi (kuo mažesnė reikšmė, tuo mažiau panašių nuomonių santraukoje)
- σ_{rep} - minimalus reprezentatyvumo slenkstis algoritmo efektyvumui užtikrinti;
- σ_{read} - minimalus skaitomumo slenkstis algoritmo efektyvumui užtikrinti;

Kaip apskaičiuoti įverčius $S_{rep}(m_i)$ ir $S_{read}(m_i)$ galima pasiskaityti [13] straipsnyje, arba galima pasirinkti bet kokias kitas funkcijas, kurios mūsų manymu gerai atspindėtų reprezentatyvumą ir skaitomumą. Priklausomai nuo pasirinktų funkcijų, priklausys ir viso algoritmo efektyvumas. Minėtame straipsnyje, vienos frazės $S_{read}(m_i)$ yra apskaičiuojamas, kaip visų trigramų toje frazėje sąlyginių tikimybių vidurkis, remiantis tuo metu prieinamo *Microsoft N-Gram service* duomenimis, o $S_{rep}(m_i)$ skaičiavimui buvo panaudotas PMI (*Pointwise Mutual Information*) rodiklis, parodantis ryšio/asociacijos stiprumą.

Algoritmas turi tris pagrindinius žingsnius:

1. **Pamatinių bigramų generacija.**
   Pirmiausia, iš gauto teksto sugeneruojame dažniausiai pasikartojančių unigramų aibę. Į šią aibę pateks visi žodžiai, kurie pasikartoja daugiau kartų negu pasikartojimų sekos mediana. Apskaičiuodami medianą, atmetame visus žodžius, kurie pasirodo tik vieną kartą. Tuomet iš šios aibės sugeneruojame visas įmanomas bigramas ir pasiliekame tik tas, kurios tenkina sąlygą - $S_{rep} >= \sigma_{rep}$

2. **Įvertintų n-gramų generacija.**
   Turėdami bigramų sąrašą, galime kiekvienai bigramai sugeneruoti aukštesnio laipsnio n-gramas, pasinaudodami algoritmu, pateiktu 4 kodo ištraukoje. Į pavaizduotą funkciją paduodame bigramą, o toliau, rekursijos pagalba gauname aibę mikro-nuomonių. Funkcijoje pirmiausia patikriname, ar paduodama frazė tenkina mūsų minimalius nustatymus (3 eilutė) ir priešingu atveju sustabdome algoritmą. Jeigu sąlygos tenkinamos, toliau tikriname, ar duotai frazei egzistuoja panaši frazė (6 eilutė), jau esanti kandidatų sąraše. Jeigu panaši frazė

(X) egzistuoja, bet jos įvertis yra mažesnis negu naujos frazės (7 eilutė), tai tuomet senąją frazę pakeičiame nauja. Jeigu duotai frazei nėra panašios frazės sąraše, tuomet pridedame frazę į kandidatų sąrašą (12 eilutė). Toliau, iš pradinio sąrašo išsirenkame bigramų poaibį, kurių pirmas žodis lygus duotos frazės (p) paskutinam žodžiui (14 eilutė). Paeiliui einame per šį sąrašą ir jeigu duota frazė nėra veidrodinis atspindys einamosios bigramos, tuomet šią frazę sujungiame su einamąja bigrama, įvertiname naujai gautą frazę (18 eilutė) ir toliau rekursiškai kartojame tą patį su gauta fraze (*newPhrase*).

3. **Mikro-nuomonės generacija.**
   Šis žingsnis yra labai paprastas - sugeneruotas mikro-nuomones iš antro žingsnio rikiuojame į santrauką, pradėdami nuo didžiausio įverčio ($S_{rep}(p) + S_{read}(p)$). Taip darome tol, kol pasiekiame savo užsibrėžtą įvertį σ_{ss}.

```
1 def GenerateCandidates(p):
2         foreach p in candidateList:
3                 if (S_read(p) < σ_read OR S_rep(p) < σ_rep) then
4                         return
5                 end if
6                 if (p ~ X) then
7                                 if S_rep(p) + S_read(p) > S_rep(X) + S_read(X)) then
8                                         candidatesList.Remove(X)
9                                         candidatesList.Add(p)
10                                end if
11                else
12                        candidatesList.Add(p)
13                end if
14                joinList = GetList(seedBigrams)
15                foreach bigram in joinList do
16                        if NotMirror(p, bigram) then
17                                newPhrase = Merge(p, bigram)
18                                Score(newPhrase)
19                                GenerateCandidates(newPhrase)
20                        end if
21                end for
22        end for
```

4 išeities kodas. Kandidatinių n-gramų generacijos pseudokodas

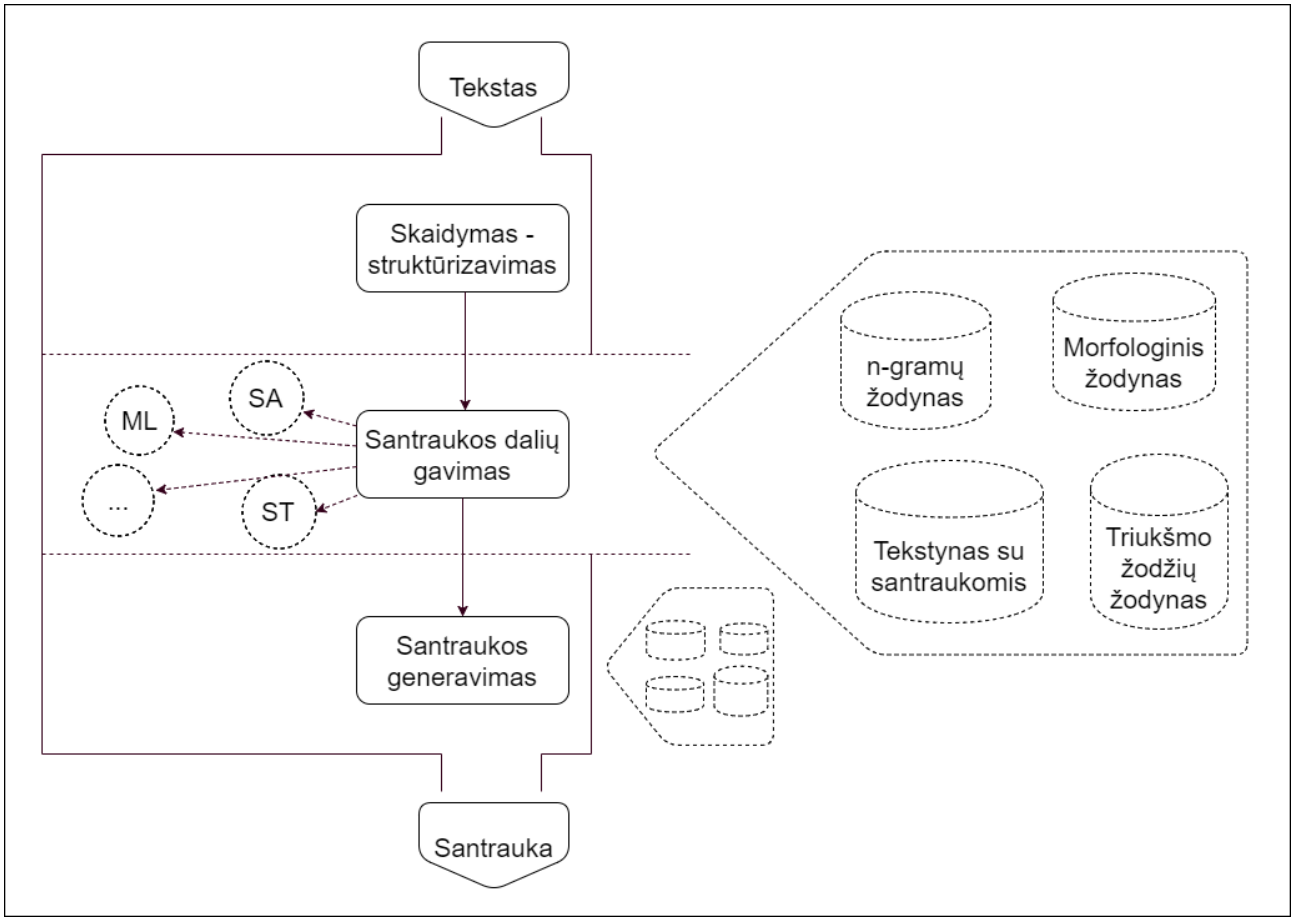
Apibendrinkime šio metodo pliusus ir minusus:

- Algoritmas generuoja koncentruotas, gramatiškai taisyklingas frazes;
- Algoritmo implementacija nėra sudėtinga;
- Godus algoritmas (*greedy algorithm*) - nėra pilnai optimalus, tačiau labai greitas ir duoda gerus rezultatus;
- Priklausomai nuo pasirinktų įverčio funkcijų, gali reikėti apmokymo.

2.5. Teksto suvedimas į santrauką

Samarizacijos procesas - tai procesas, konvertuojantis turimą tekstą (šaltinį) į trumpesnę jo kopiją, pateikiant informaciją koncentruotai ir išlaikant turinio esmę nepakitusią. Šis procesas gali

grąžinti dviejų tipų santraukas - orientacines arba informatyviąsias. Pirmojo tipo santraukos yra tarsi nuoroda skaitytojui, kur reikėtų atkreipti dėmesį tekste, arba pagalba rašančiajam santrauką. Tai nėra pilnas tekstas ir skaitant vien tik tokią santrauką, gali būti neaišku, apie ką yra tekstas. Antrojo tipo santraukos yra išbaigtas tekstas, kuris turėtų pakeisti teksto santrauką - tai pilnais sakiniais išreikštos mintys. Skaitant tokią santrauką, neskaičius originalaus teksto, turėtų būti pilnai aišku, ką norima pasakyti tuo tekstu.



8 pav. Bendrinė diagrama teksto suvedimui į santrauką.

Kad ir kokią santrauką bandytume gauti, paprastai samarizacijos metodas turi tuos pačius žingsnius - skiriasi tik būdai, kuriais įgyvendiname tuos žingsnius. Kaip pavaizduota 8 paveikslėlyje, yra trys pagrindiniai žingsniai suvesti tekstą į santrauką:

1. Šaltinio struktūrizacija ir (arba) skaidymas - suvedimas į tokią formą, kurią suprastų mūsų metodas. Tai gali būti teksto paragrafų atskyrimas, sakinių ir žodžių atskyrimas, ilgo teksto skaidymas į mažesnes dalis. Taip pat, šiame žingsnyje gali būti vykdomas teksto apvalymas - šalinamos tam tikros teksto dalys, keičiami žodžiai, skyrybos ženklai ir panašiai.

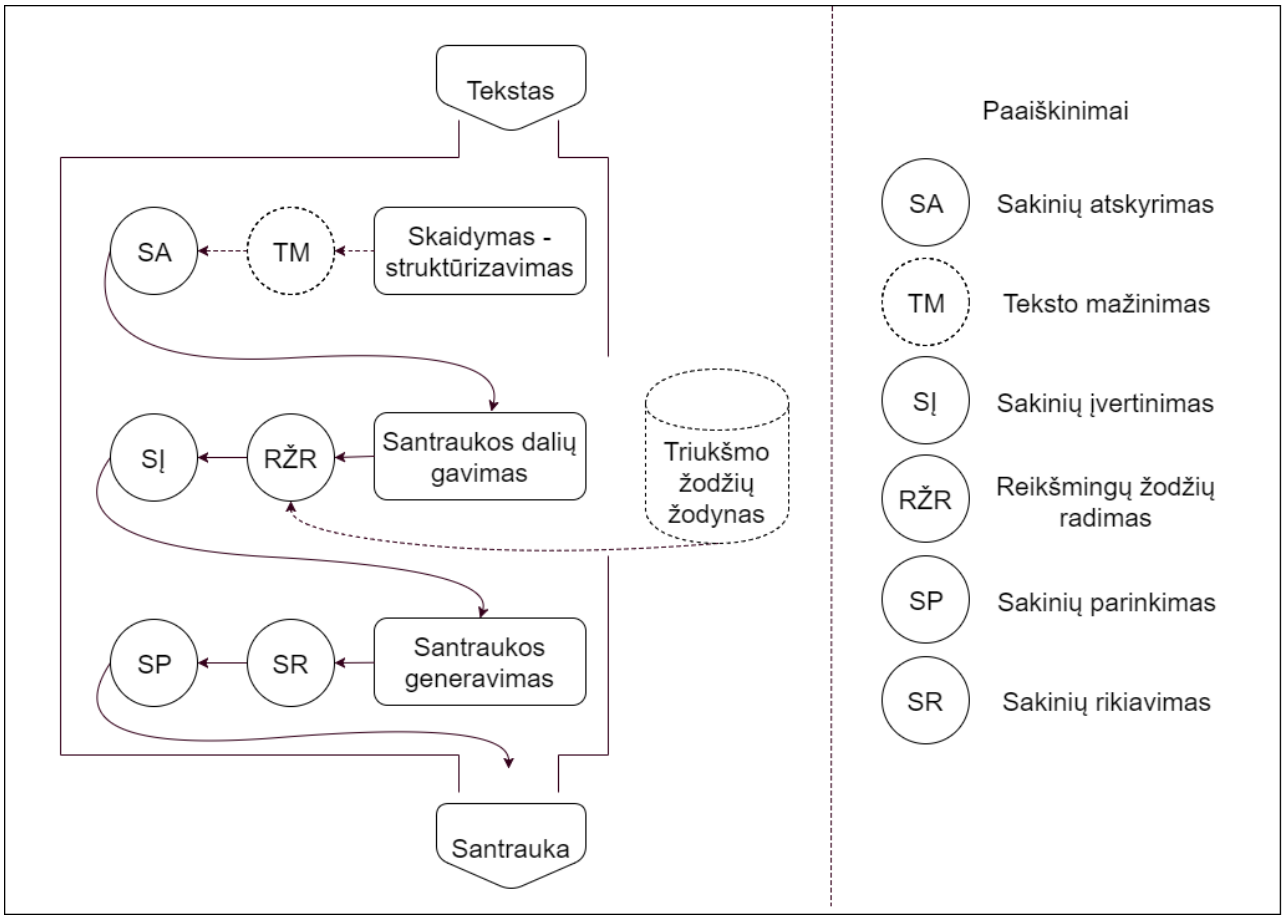
2. Santraukos dalių gavimas - šis žingsnis dažniausiai yra esminis visuose metoduose. Paprastai šioje dalyje yra gaunamos tam tikros teksto dalys, nurodančios teksto esmę, čia yra išskiriamas kontekstas. Galima sakyti, kad po šio žingsnio gauname orientacinę teksto santrauką - tai dar nėra pilni sakiniai ir gali būti neišsamu, tačiau, skaitytojas jau žinotų, į kurias vietas tekste koncentruoti savo dėmesį, arba kurias dalis būtina paminėti santraukoje.

3. Santraukos generavimas - šis žingsnis užtikrina, kad gauta santrauka bus skaitoma ir suprantama skaitytojui. Čia imame antrame žingsnyje gautą rezultatą ir jį pertvarkome taip, kad

būtų patogu skaityti, tai yra, raktažodžius ir frazes pertvarkome į pilnus sakinius.

8 paveikslėlyje pavaizduotoje diagramoje, punktyrinė linija žymi metodo dalis, kurios nėra privalomos, tačiau gali būti naudojamos antrame žingsnyje. Priklausomai nuo paduodamų išteklių (dešinėje pusėje punktyru apibrėžta figūra), galime įvesti sintaksinę analizę (SA), mašininį mokymą (ML nuo "Machine Learning" angl.), statistinę analizę ar kitą mūsų manymu tinkantį metodą. Atitinkamai, šiuos metodus galime papildomai pritaikyti paskutiniame žingsnyje, norėdami pagerinti gautos santraukos kokybę - perrašyti sakinius į gramatiškai taisyklingus, perfrazuoti arba papildomai sutrumpinti.

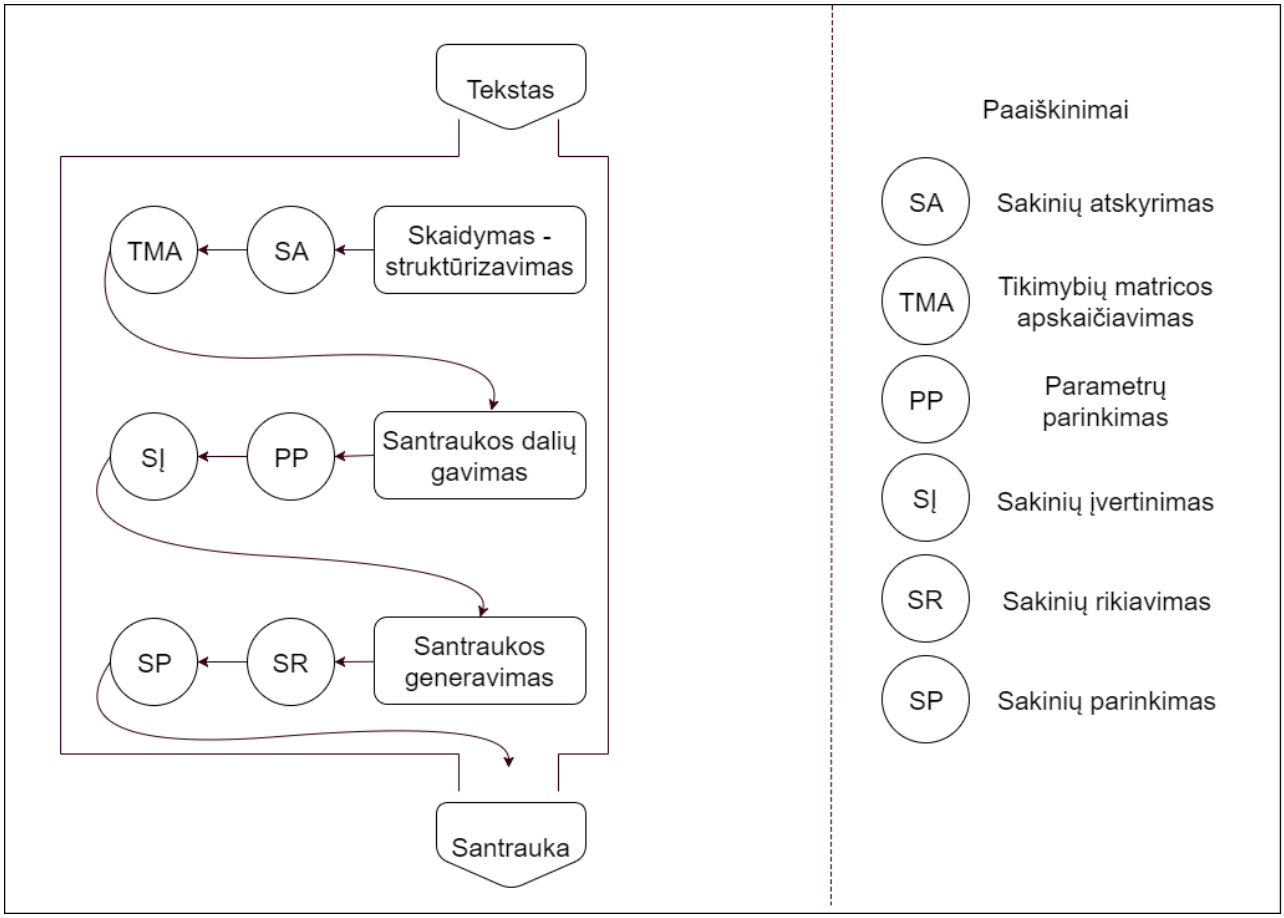
Du prieš tai ėję poskyriai aprašo tarpusavyje skirtingus samarizacijos metodus. Pavaizduosime juos per 8 diagramos išraišką.



9 pav. H. P. Luhn metodo diagrama.

9 paveikslėlis vaizduoja 2.3.1 skyrelyje aprašytą metodą teksto santraukai gauti. Pirmiausia turimas tekstas yra išskirstomas į mažesnius tekstus, jeigu šaltinis yra ganėtinai didelis. Tuomet kiekvienas iš tų tekstų atskirai yra skaidomi sakiniais ir paduodami į antrąjį žingsnį. Čia atliekame statistinę analizę ir atrenkame reikšmingus žodžius - galime pasinaudoti triukšmo žodžių žodynu, kad atskirtume nereikšmingus žodžius, arba galime naudotis viršutine ir apatine ribomis, pavaizduotomis 6 paveikslėlyje. Sudarę reikšmingų žodžių sąrašą, pritaikome 2.2 formulę kiekvienam sakiniui, kad gautume jo įvertį tekste. Šiuos įverčius perduodame į trečiąjį žingsnį, kur sakiniai yra išrikiuojami pagal įvertį mažėjimo tvarka. Toliau, nuo pradžios pasirenkame tam tikrą sakinių skaičių, kurie sudarys teksto santrauką. Sakinių skaičius priklausys nuo metodo implementacijos - jei norime, galime leisti vartotojui pasirinkti šį skaičių, arba parinkti tam tikrą procentą sakinių nuo viso teksto, kuris mūsų manymu atitiktų santrauką. Jeigu pirmajame žingsnyje atlikome teksto

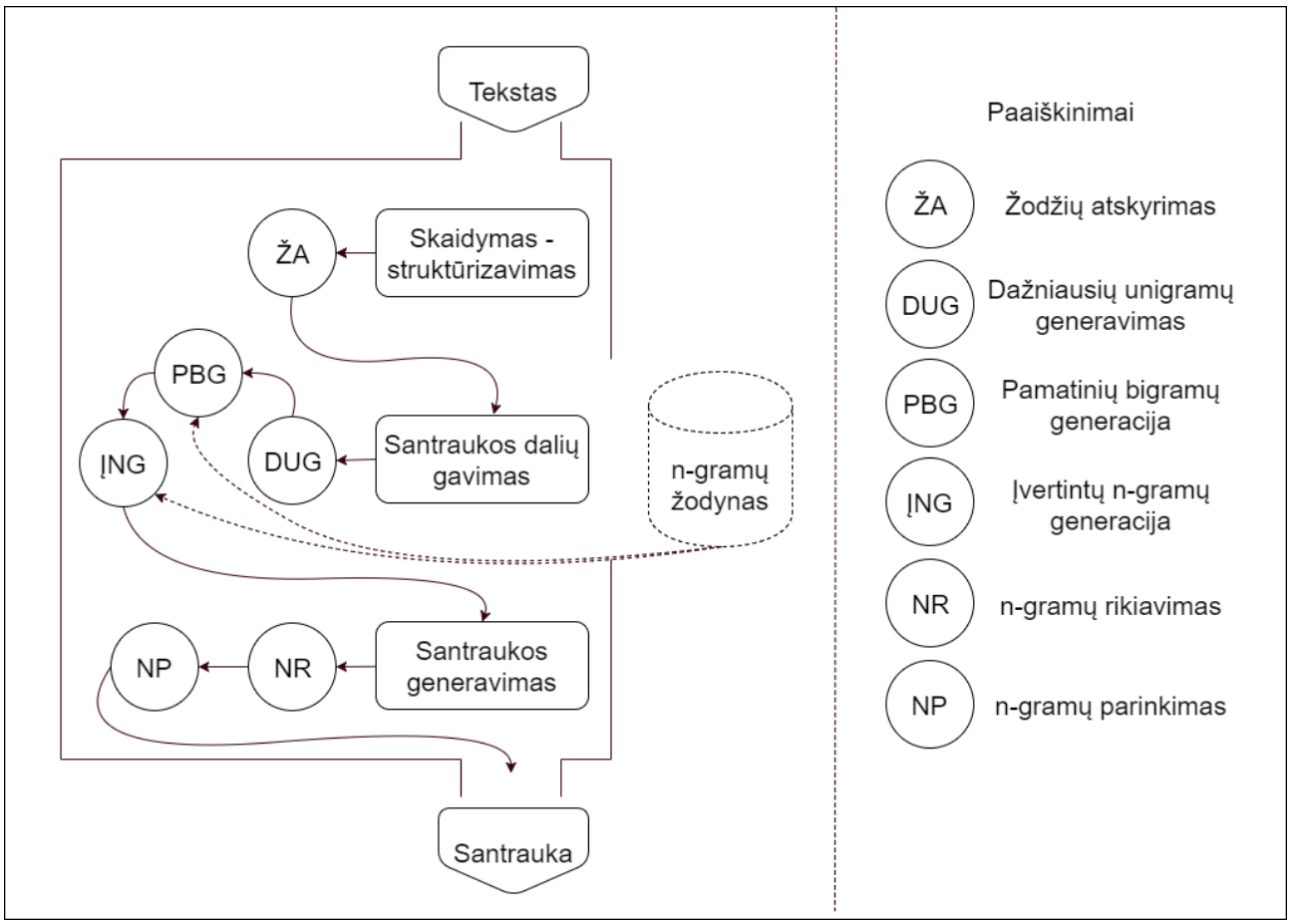
mažinimą - išskaidėme tekstą į mažesnius tekstus, tuomet antras ir trečias žingsniai yra kartojami kiekvienai teksto daliai, o gautos santraukos yra atitinkamai sujungiamos į vieną.



10 pav. *TextRank* metodo diagrama.

Toliau, 10 diagramoje vaizduojamas 2.3.2 poskyryje aprašytas *TextRank* metodas. Jis savo eiga labai panašus į H. P. Luhn metodą - pirmiausia įvertiname kiekvieną sakinį tekste, tuomet tuos sakinius išrikiuojame nuo didžiausio įverčio iki mažiausio, ir galiausiai parenkame kažkiek sakinių nuo pradžios, kurie sudarys santrauką. Šio metodo atveju, skirsis logika, kaip apskaičiuojami įverčiai. Taigi, po išskaidymo sakiniais, pirmame žingsnyje apskaičiuojame tikimybių matricą, kuri bus naudojama sekančiame žingsnyje. Antrame etape, parenkame parametrus eps ir d, kurie gali būti parinkti kaip konstantos arba priklausyti nuo teksto pobūdžio. Šiuos parametrus, drauge su tikimybių matrica atiduodame į 3 kodo ištraukoje parodytą algoritmą ir apskaičiuojame kiekvieno sakinio įvertį. Gautus įverčius sekančiame žingsnyje išrikiuojame mažėjimo tvarka ir atsirenkame tam tikrą skaičių sakinių su didžiausiais įverčiais.

Mikro-nuomonių generavimas labiausiai išsiskiria iš prieš tai nagrinėtų santraukos generacijų. Pagrindinis šio metodo išskirtinumas yra tas, kad nenagrinėjame atskirai kiekvieno sakinio, o gauta santrauka nėra tik sakiniai, tiesiogiai paimti iš teksto. Pirmajame žingsnyje atskiriame tik žodžius, kad galėtume išskaičiuoti jų pasikartojimus. Toliau, antrame žingsnyje randame dažniausias unigramas, kitaip sakant randame populiariausius žodžius tekste. Kaip aprašyta 2.4.1 skyrelio, antrame algoritmo žingsnyje, toliau iš šių unigramų sudarome pamatines bigramas, kurios atitinka reprezentatyvumo sąlygą. Iš šios gautos bigramų aibės, lipdome aukštesnio laipsnio n-gramas, pagal 4 kodo ištraukoje pavaizduotą algoritmą. Taip sugeneruojame naują aibę n-gramų. Trečiajame žingsnyje išrikiuojame turimas n-gramas pagal jų įverčius nuo didžiausio iki mažiausio. Iš šios



11 pav. Mikro-nuomonių generavimo metodo diagrama.

sekos paimame tiek n-gramų, kad jų bendras ilgis neperliptų iš anksto užsibrėžto ilgio parametro. Gauta seka ir bus mūsų santrauka.

2.6. Matematinis modelis

Norėdami aiškiai ir tiksliai aprašyti matematinį modelį, turime įsivesti žymėjimus, kuriuos vėliau naudosime.

Trumpumo dėlei, įsivesime žymėjimus įvairiems žodynams ir kitoms aibėms:

- Oficialus Lietuvių kalbos žodynas:

$$\mathbb{LKZ} ::= \{lkz_1, lkz_2, ..., lkz_i\}, \text{kur } i \in \mathbb{N}, \text{ o } lkz_i - \text{žodis iš žodyno;} \tag{2.8}$$

- Vardų žodynas - šią aibę sudarys visi žodžiai, kurie žymi vardus ir pavadinimus. Jie gali taip pat egzistuoti ir $\mathbb{LKZ}$ aibėje, tačiau nebūtinai. Į šią aibę taip pat papuola visi ne lietuviški vardai ir pavadinimai:

$$\mathbb{V} ::= \{v_1, v_2, ..., v_i\}, \text{kur } i \in \mathbb{N}, \text{ o } v_i - \text{bet koks vardas arba pavadinimas;} \tag{2.9}$$

- Trumpinių žodynas - šios aibės nariai gali būti įvairūs oficialūs trumpiniai, kaip "EU" arba "UK", o taip pat, naujai sudaryti trumpiniai, kurių prasmė numanoma tik iš konteksto. Tokie trumpiniai paprastai nėra randami žodyne.

$$\mathbb{TR} ::= \{tr_1, tr_2, ..., tr_i\}, \text{kur } i \in \mathbb{N}, \text{ o } tr_i - \text{bet koks trumpinys;} \tag{2.10}$$

POSn	POS	POSn	POS
0	Neatpažintas žodis	9	Jungtukas
1	Daiktavaris	10	Jaustukas
2	Būdvardis	11	Ištiktukas
3	Skaitvardis	12	Vardas Pavarde
4	Įvardis	13	Vardas arba Pavadinimas
5	Veiksmažodis	14	Trumpinys
6	Prieveiksmis	15	Dalyvis
7	Dalelytė	16	Padalyvis
8	Prielinksnis		

1 lentelė. Kalbos dalių sąrašas su numeracija.

aukšč.l.	vyr.g.	vns.	Š.	tar.nuos.	es.l.	neveik.dlv.
nelyg.l.	įv.f.	dgs.	V.	ties.nuos.	bus.l.	veik.dlv.
lygin.l.	neįv.f.	K.	Vt.	1asm.	būt.k.l.	nežym.įv.
bev.g.	sangr.f.	N.	G.	2asm.	es.l.	sant.įv.
mot.g.	nesangr.f.	Įn.	liep.nuos.	3asm.	infinityvas	atsk.įv.

2 lentelė. Kalbos dalių ypatybių sąrašas.

- Skirtukų aibė - į šią aibę patenka visi lietuvių kalboje sutinkami skyrybos ženklai ir jų deriniai;

$$\mathbb{SK} ::= \{sk_1, sk_2, ..., sk_i\}, \text{kur } i \in \mathbb{N}, \text{ o } sk_i - \text{bet koks skirtukas ar skirtukų derinys;} \quad (2.11)$$

- Kalbos dalių aibė - į šią aibę patenka visos lietuvių kalbos dalys;

$$\mathbb{KD} ::= \{kd_1, kd_2, ..., kd_i\}, \text{kur } i \in \mathbb{N}, \text{ o } kd_i - \text{bet kokia kalbos dalis.} \quad (2.12)$$

4 apibrėžimas. Kalbos dalimi kd_i vadinsime ypatybių rinkinį $< POSn, POS, y_1, y_2, ..., y_m >$, kur $POSn$ žymi kalbos dalies skaitinę išraišką, POS - kalbos dalies pavadinimą (žiūrėti 1 lentelę), o y_m - kuriai nors kalbos daliai būdingą ypatybę (žiūrėti 2 lentelę). Kiekviena ypatybė šiame rinkinyje pasikartoja tik kartą, bet ta pati ypatybė gali atsirasti keliose skirtingos kalbos dalyse. Pavyzdžiui - *<1,daiktavardis,vyr.g.,N.,dgs.>, <2,būdvardis,bev.g.,V.,vns.,nelyg.l.,neįv.f.>* ir panašiai.

Remdamiesi aukščiau pateiktais žodynų apibrėžimais, galime suformuluoti mūsų darbe naudojamo žodyno apibrėžimą:

5 apibrėžimas. Žodynu vadinsime sąjungą mažesnių žodynų:
$\mathbb{Z} = \mathbb{LKZ} \cup \mathbb{V} \cup \mathbb{TR} ::= \{z_1, z_2, ..., z_i\}$, kur $i \in \mathbb{N}$, o z_i žymės bet kurį sutinkamą žodį iš bet kurio žodyno arba žodynų.

6 apibrėžimas. Žodžiu vadinsime porą, sudarytą iš žodžio $z_i \in \mathbb{Z}$ ir jam priskiriamos kalbos dalies kd_m:
$\bar{z_i} ::= (z_i, kd_m)$, čia $i, m \in \mathbb{N}$.

7 apibrėžimas. Sakiniu S vadinsime seką žodžių $\bar{z}_i$ su jiems priskiriamais skirtukais $sk_i \in \mathbb{SK}$ (skirtukas gali būti ir tuščias simbolis):
$S ::= \{< \bar{z}_i, sk_i >\} ::= \{< (z_i, kd_m), sk_i >\}$, kur $i, m \in \mathbb{N}$

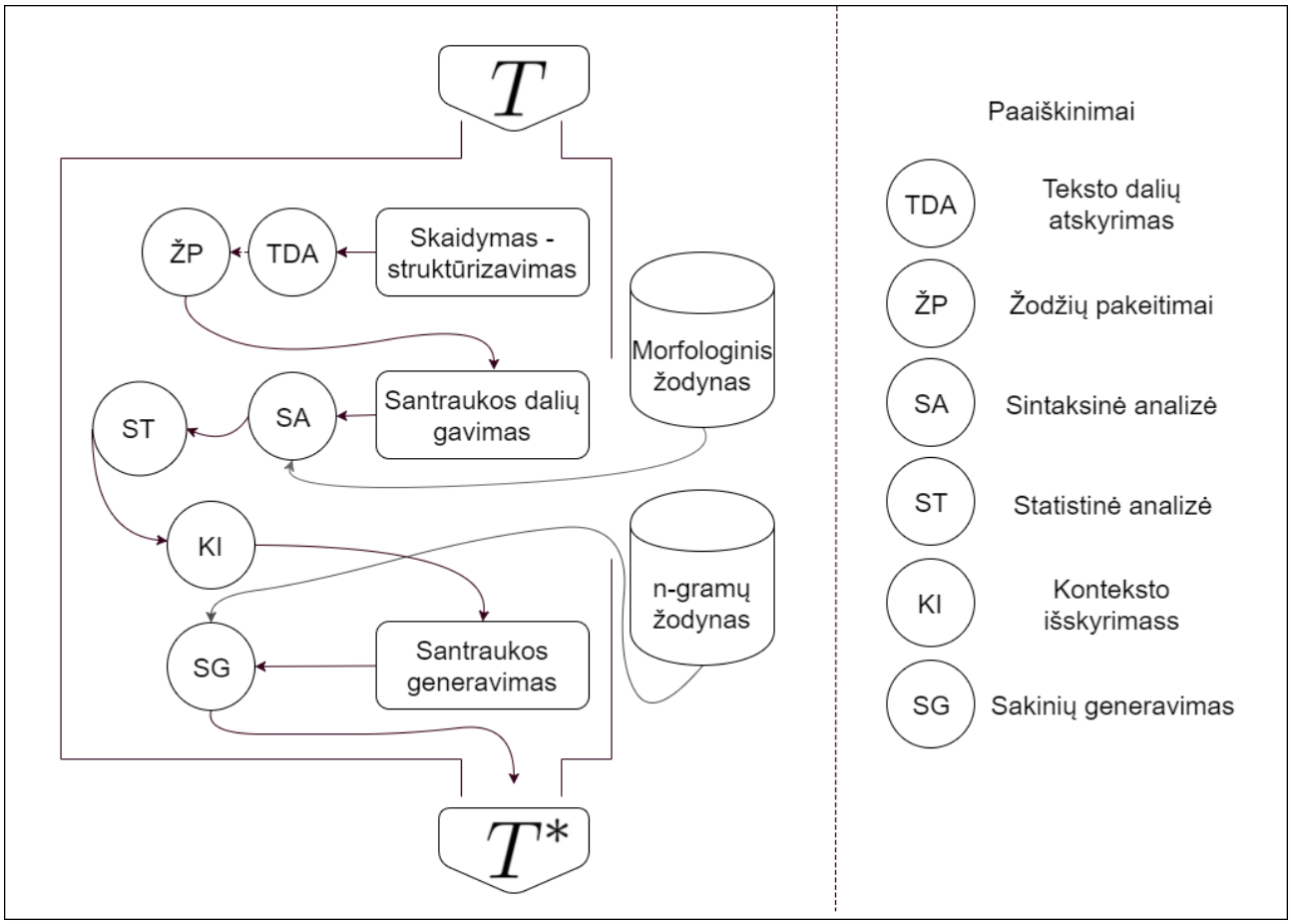
8 apibrėžimas. Paragrafu vadinsime seką sakinių: $P ::= \{S_1, S_2, ..., S_i\}$, kur $i \in \mathbb{N}$

9 apibrėžimas. Tekstu vadinsime seką paragrafų: $T ::= \{P_1, P_2, ..., P_i\}$, kur $i \in \mathbb{N}$

10 apibrėžimas. Teksto $T ::= \{P_1, P_2, ..., P_n\}$ santrauka arba samarizacija vadinsime naują seką, naujai sudarytų sakinių $T_i^* ::= \{P_1^*, P_2^*, ..., P_l^*\}$ kokiam nors l ir $n \in \mathbb{N}$, kur $l <= n$.

11 apibrėžimas. Santraukos funkcija vadinsime tokią funkciją, kuri duotą tekstą T suveda į santrauką T^*. Tokią funkciją žymėsime:

$$\mathcal{S}(T) ::= T^* \tag{2.13}$$



12 pav. Teksto suvedimo į santrauką diagrama.

12 paveikslėlyje pavaizduota diagrama, pagal kurią duotas tekstas suvedamas į santrauką. Ši diagrama nubraižyta remiantis schema, pateikta 8 paveikslėlyje. Norėdami sukurti metodą, kuris galėtų sugeneruoti santrauką, kuo artimesnę žmogiškajai santraukai, naudojamės ankstesnių autorių idėjomis bei įvedame sintaksinę analizę. Žemiau visi žingsniai yra paaiškinami plačiau:

1. **Pradinio teksto paruošimas.** Turimas tekstas T yra paruošiamas sintaksinei analizei - suvedamas į tokią formą, kokios tikisi algoritmas. Čia atliekami įvairūs teksto valymo darbai, pakeitimai ir pašalinimai. Nustatomos paragrafų, sakinių ir žodžių ribos.

Sakykime, turime tekstą T, kurį norime suvesti į santrauką T^*. Jeigu šį tekstą įsivaizduosime, kaip raidžių, skaičių, tarpų ir kitų simbolių seką, tai jį galime apibrėžti šitaip:

$$T = \{t_1, t_2, ..., t_n\}, kurt_n \in (aA,ąĄ,...,žŽ,0,1,...,9) \cup \mathbb{SK}. \tag{2.14}$$

Šią seką suskirstome į nesikertančius posekius, kurie žymės atskirus paragrafus. Kiekvieną paragrafą sunumeruojame. Atitinkamai, tuos paragrafus išskiriame į mažesnius ir taip pat nesikertančius posekius, kurie žymės sakinius, ir sunumeruojame.

Toliau sakinius išskaidome į žodžių sekas ir kiekvieną žodį taip pat sunumeruojame. Žodžius išsaugome masyve arba lentelėje, drauge su paragrafo, sakinio ir žodžio eilės tvarka bei visais skyrybos ženklais, einančiais po šio žodžio. Čia neužtenka turėti vien tik žodžių tekste tvarką, bet svarbu ir žinoti, kurie žodžiai priklauso kuriems sakiniams, ir kokia tvarka tie sakiniai išsidėstę paragrafe. Ignoruodami sakinių ar paragrafų išsidėliojimą originaliame tekste, galime iš esmės pakeisti teksto prasmę. Skyrybos ženklai, priklausantys kažkuriam žodžiui, bus svarbūs sintaksinei analizei.

Toliau reikia surasti kiekvieno žodžio kalbos dalį (arba dalis), bei jai priskiriamas savybes kaip minėta 4 apibrėžime. Tokiu būdu, turėtus teksto žodžius pakeičiame poromis, kuriose yra pats žodis drauge su jam priskiriama kalbos dalimi. Beje, ta kalbos dalis gali būti išreikšta sąrašu iš kelių kalbos dalių. Taigi, turimą tekstą T suvedame į žemiau pavaizduotą pavidalą:

$$T -> T' = \left\{ \begin{array}{c} P_1 \\ P_2 \\ ... \\ P_n \end{array} \right\} = \left\{ \begin{array}{cccc} S_{11} & S_{12} & ... & S_{1k_1} \\ S_{21} & S_{22} & ... & S_{2k_2} \\ ... & ... & ... & ... \\ S_{n1} & S_{n2} & ... & S_{nk_n} \end{array} \right\} \tag{2.15}$$

Naudodamiesi S apibrėžimu (7), T' suvedame į tokį pavidalą:

$$T -> T' = \left\{ \begin{array}{c} \{< (z_i, kd_i), sk_i > | 0 < i <= m_1\} \\ \{< (z_i, kd_i), sk_i > | m_1 < i <= m_2\} \\ ... \\ \{< (z_i, kd_i), sk_i > | m_n < i <= m_{n+1}\} \end{array} \right\} = \{< (z_i, kd_i), sk_i >\} \tag{2.16}$$

Čia $n, k, i, m \in \mathbb{N}$, $m_1 < m_2 < ... < m_{n+1}$.

2. **Santraukos dalių gavimas**. Šioje dalyje pritaikome sintaksinę analizę, randame pagrindines sakinio dalis - veiksnį, tarinį. Taip pat, atliekame statistinę analizę - randame teksto raktažodžius. Toliau išskiriame kontekstą sujungdami sintaksinės ir statistinės analizės rezultatus. Tokiu būdu randame pagrindinius viso teksto veikėjus, bei veiksmus, susijusius su jais.
Remdamiesi 3.3 skyrelyje sudarytu veiksnio ir tarinio aprašu, galime rasti kiekvieno sakinio centrą (arba centrus, jeigu sakinys ne vientisinis). Turėdami tokias sakinių ištraukas, jau galėtume sakyti, kad turime dalį konteksto, kadangi pagrindinė mintis ir turėtų suktis apie

veikėją ir jo daromus veiksmus. Deja, vien tik to nepakanka, kadangi ne visi veiksmai yra svarbūs teksto idėjai, ir ne visi veikėjai yra pagrindiniai. Dėl to, gavę sakinio centrus, dar turime atsirinkti, kuriuose centruose dalyvauja mūsų pagrindinis veikėjas. Darysime prielaidą, kad pagrindinis veikėjas arba veikėjai bus tas veiksnys, kuris sutinkamas dažniausiai ir taip pat yra minimas dažniausiai kitose sakinio dalyse.
Kitas ženklas, nurodantis pagrindinę teksto mintį, yra žodžių pasikartojimai - kuo dažniau kartojasi tas pats žodis ar jo formos, tuo labiau tikėtina, kad tas žodis žymi svarbią detalę. Taigi, pirmiausia rasime teksto raktažodžius. Čia kyla klausimas, kokie žodžiai turėtų patekti į raktažodžių aibę - remdamiesi [13] šaltinio autorių pasiūlymu, dažniausiomis unigramomis arba raktažodžiais laikysime, tuos žodžius, kurie pasikartoja daugiau kartų negu visų žodžių pasikartojimų mediana. Aiškumo dėlei, šį teiginį suformuluosime žemiau.

12 apibrėžimas. Sakykime, kad turime išvardintus visus teksto žodžius, drauge su jų pasikartojimų sumomis. Į tuos pasikartojimus prisumuojame ir sinonimus, ir žodžių formų pasikartojimus. Taip pat praleidžiame visus žodžius, kurie pasikartojo tik vieną kartą. Tokiu būdu, gauname skaičių seką $P = \{p_1, p_2, ..., p_n\}$, kur $n \in \mathbb{N}$ ir kuriai galime paskaičiuoti medianą. Tuomet, teksto raktiniais žodžiais vadinsime tuos žodžius, kuriems galioja nelygybė:

$$COUNT(z_i^*) > MEDIANA(P), \text{ kur } z_i^* \in (Z), \text{o } i \in \mathbb{N} \tag{2.17}$$

Be to, randame pagrindinį teksto veikėją arba veikėjus, imdami sankirtą raktinių žodžių su viso teksto sakinių veiksniais:

13 apibrėžimas. Tarkime, kad aibė $V(T)$ yra visų teksto sakinių veiksnių aibė, o aibė $K(T)$ - raktažodžių aibė. Tuomet pagrindiniu teksto veikėju vadinsime sankirtą:

$$M(T) = V(T) \cap K(T); \tag{2.18}$$

Lygindami $M(T)$ rezultatus su veiksnių - tarinių poromis, galime surasti visus veiksmus susijusius su pagrindiniu veikėju (arba veikėjais). Tokiu būdu gauname aibę frazių, teiginių ir sakinių iš teksto, kurie atspindi teksto esmę. Be to, paliekame tik unikalius teiginius, pašalindami pasikartojančius.

3. **Santraukos generavimas.** Iš prieš tai buvusio žingsnio gauname frazes, nedidelius sakinius, kurie nurodo teksto pagrindinę mintį. Šioje dalyje turime perdaryti šias frazes į rišlius lietuviškus sakinius bei pateikti rezultatą skaitytojui.

   Čia galime pasinaudoti žodžių n-gramomis - tikrindami kiekvienos frazės bigramas, trigamas, 4-gramas specialiame n-gramų žodyne, galime patikrinti, ar žodžių junginiai yra vartojami bei rasti tinkamesnes žodžių formas tiems junginiams. Kokio laipsnio n-gramas reikėtų tikrinti turi priklausyti nuo turimo žodžių junginio - kuo ilgesnis junginys, tuo aukštesnio laipsnio n-gramas galime tikrinti.

3. Įgyvendinimas

Šiame skyriuje pademonstruosime 2.6 skyrelyje išdėstyto modelio įgyvendinimą. Papasakosime apie problemas, su kuriomis susidūrėme, įgyvendindami metodą, kaip jas sprendėme. Anksčiau minėti žingsniai bus smulkiai aprašyti bei bus pateikta keletas pavyzdžių. Galiausiai, pateiksime rezultatus ir išvadas, gautas iš metodo įgyvendinimo.

3.1. Darbinių duomenų aprašymas

Pirmiausiai, norėtume aprašyti tekstus, kuriuos nagrinėjome ir kuriais rėmėmės sudarydami metodą. Tai yra nedidelio ilgio (1-3 puslapių) viešai prieinami straipsniai. Jie gali būti parašyti tiek ilgamečio profesionalo, tiek mėgėjo, dėl to neatmetama gramatinių klaidų tikimybė. Šiuose straipsniuose nėra tiesioginės kalbos, tai yra autoriaus nuomonės išreiškimas, apie vienokią ar kitokią situaciją. Šis metodas taip pat galėtų būti pritaikomas ilgesniems straipsniams negu nagrinėjome, tačiau nebūtų tinkamas (arba mažai tinkamas) moksliniams, medicininiams straipsniams ar grožinei literatūrai.

3.2. Duomenų paruošimas

Kaip ir bet kurioje kitoje užduotyje, duomenų apdirbimas pradedamas nuo jų paruošimo - turimus duomenis turime konvertuoti į tokį formatą, kuris būtų lengvai suprantamas ir interpretuojamas programuojant. Turint omenyje, kad pradinis tekstas tėra didelis raidžių ir kitų simbolių, supuolančių į žodžius, kratinys, pirmiausia nusistatome ribas, kurios žymės žodžio pradžią ir pabaigą, sakinio pradžią ir pabaigą, bei paragrafo pradžią ir pabaigą.

Savo programoje, duomenų nuskaitymą bei dalinį apdirbimą darysime *Python* programavimo kalba. Tekstiniame faile viena eilutė reiškia vieną paragrafą, todėl nėra sudėtinga atskirti - atskyrimas įvyksta jau nuskaitant failą. Nuskaitytos eilutės yra sunumeruojamos ir dedamos į masyvą. Eilučių tvarka yra labai svarbi, kadangi viename paragrafe pradėta dėstyti mintis, gali būti užbaigiama kitame paragrafe, o taip pat, gali būti, kad antrame paragrafe bus pradedama nauja mintis.

Toliau iš kiekvieno paragrafo reikia išskirti atskirus sakinius. Tai atrodytų nesudėtinga problema - sakinius skiria taškai, klaustukai, šauktukai. Nesudėtingiems sakiniams tokia prielaida yra teisinga ir tai galima nesunkiai įgyvendinti programatiškai. Tačiau yra ir tokių situacijų, kur sakinys baigiasi pavyzdžiui skliaustu - "Labai mažai žinių pateikiama apie lietuvių kalbos priegaides. (Apskritai į tarties dalykus Jablonskis kreipė mažai dėmesio.)" [14]. Kitas dažnai sutinkamas pavyzdys yra taškai, skiriantys ne sakinius, o trumpinius ar vardo pirmą raidę - "Apskritai į tarties dalykus J. Jablonskis kreipė mažai dėmesio." Galima rasti nemažai implementacijų įvairioms programavimo kalboms, kurios geba atskirti ir sudėtingesnius sakinius iš paragrafo, tačiau dauguma tokių implementacijų yra sukurtos anglų ar kitoms užsienio kalboms, ir dėl to, gali pilnai neveikti lietuvių kalbai.

Sakinių atskyrimui atlikti naudosime Python kalbos paketą - *segtok*. Šis paketas skirtas Indo-Europiečių kalbų tekstų atskyrimui į atskirus sakinius. Tiesa, jis buvo suprojektuotas remiantis ispanų, anglų ir vokiečių kalbomis, todėl naudojant su kitomis kalbomis, gali atsirasti tokių situacijų, kur sakiniai bus išskirti neteisingai. Mūsų atveju, problema iškyla su minėtu sakiniu - "Apskritai į tarties dalykus J. Jablonskis kreipė mažai dėmesio." Programa nesupranta, jog "J. Jablonskis" tai yra vardo pirma raidė ir pavardė, o ne naujo sakinio pradžia, todėl vietoje vieno sakinio gauname du. Sakinys yra interpretuojamas teisingai, jeigu toks trumpinys eina sakinio pradžioje arba jeigu

taškas naudojamas skaičiaus dešimtainei išraiškai žymėti. Programos paketas taip pat atpažįsta trumpinius, kurie atskirti taškais be tarpų (pvz.: "A.R.S.") Minėtai problemai spręsti, galimi keli būdai:

1. Pakeisti visus trumpinius jų atitikmenimis. Pagrindinė problema su kuria tokiu atveju susidurtume - trumpinių žodyno sudarymas. Toks žodynas nebūtų galutinis, kadangi trumpiniai nėra patvirtinami kalbininkų - kiekvienas galime sugalvoti savo trumpinį ir jį naudoti. Žinoma, trumpiniai gali būti "atspėjami" pagal kontekstą (jeigu atitikmuo naudojamas tame pačiame tekste), tačiau tai irgi yra sudėtinga problema, reikalaujanti daug papildomų resursų.

2. Pašalinti visus trumpinius, kuriuos sudaro tik viena raidė ir taškas, iš teksto. Tokio trumpinio pašalinimas nepakeis teksto prasmės, nes jis neneša jokios naudingos informacijos - ar sakytume "Jablonskis", ar "J. Jablonskis" vis tiek suprastume, apie ką kalbama.

3. Ignoruoti tokius sakinių atskyrimo netikslumus ir tikėtis, kad tokių atvejų nebus daug ir stipriai nepaveiks galutinio rezultato.

Mūsų atveju pasirenkame ignoruoti tokius netikslumus, kadangi bet kuris kitas variantas reikalauja daug pastangų, o papildomos naudos atneš nedaug. Be to, tokių netikslumų nebus daug. Taip pat, dalį sakinių atskyrimo klaidų galime ištaisyti ir po sakinių atskyrimo. Taigi, atskiriame sakinius ir juos taip pat sunumeravę dedame į masyvą.

Sekantis žingsnis - išskaidyti sakinius į žodžius. Tai nėra sudėtinga problema, jei apsibrėžiame, kas tai yra žodis teksto kontekste. Žodis bus (šiuo atveju lietuvių kalbos) abėcėlės komponenčių seka, kurios pradžią nurodo tarpas, skyrybos ženklas arba naujos eilutės simbolis, o pabaigą - skyrybos ženklas arba tarpas. Žinodami žodžių atskyrimo taisykles, jas galime perrašyti į Python *Regular Expression* šablonus ir tokiu būdu, gana lengvai suvesti sakinių masyvą į žodžių masyvą. Tiesa, kad palengvintume darbą tolesniuose žingsniuose, prieš skaidydami sakinius į žodžius, padarome tokius pakeitimus:

- Trumpiniuose pakeičiame tarpus ir taškus apatiniais brūkšneliais ("_"), bei pridedame priekyje "__TRUMPINYS_", or gale - "__". Pavyzdžiui:
  - "T.R.M.P" -> "__TRUMPINYS_T_R_M_P__"
  - "T. R. M. P." -> "__TRUMPINYS_T_R_M_P__"
  - "TRMP" -> "__TRUMPINYS_TRMP__"
- Vardo pirmos raidės ir pavardės trumpiniuose taip pat pakeičiame tarpus ir taškus apatiniais brūkšneliais, bei pridedame priekyje "__VPAVARDE_", o gale - "__". Pavyzdžiui:
  - "V. Pavardė" -> "__VPAVARDE_V_Pavardė__"
  - "V. V. Pavardė" -> "__VPAVARDE_V_V_Pavardė__"
- Vardus ir pavadinimus, einančius kartu, sujungiame į vieną žodį, pakeičiame tarpus apatiniais brūkšneliais, pridedame priekyje "__VARDAS_", o gale "__". Pavyzdžiui:
  - "Pavadinimas" -> "__VARDAS_Pavadinimas__"
  - "Juntinė Karalystė" -> "__VARDAS_Jungtinė_Karalystė__"

```
1 įvard.-daikt., V., dgs., sant. įv., mot.g.
2 įvard.-daikt., K., vns., sant. įv., mot.g.
3 daiktavardis, mot.g., V., dgs.
4 daiktavardis, mot.g., K., vns.
5 daiktavardis, mot.g., Š., dgs.
```

13 pav. Žodžio "kurios" kalbos dalių aprašymas gautas iš *www.morfologija.lt*

Atlikę minėtus pakeitimus, užtikriname, kad mūsų trumpiniai nebus išskaidyti į atskiras raides, kad vardo pirmoji raidė eis kartu su pavarde, o kelių žodžių pavadinimai nebus išskaidyti į atskirus žodžius. Tai reikalinga, nes trumpiniai išskaidyti į atskiras raides neturi jokios vertės, kaip ir vardo pirmoji raidė atskirta nuo pavardės. Kelių žodžių pavadinimus taip pat traktuojame kaip vieną žodį, nes tie žodžiai yra prasmingi tik kartu, o ne atskirai - taip išvengsime netikėtų metodo rezultatų. Kai pavadinimas sudarytas tik iš vieno žodžio, mes vis tiek norime pabrėžti, kad tai yra pavadinimas ir kad šis pavadinimas tikriausiai nebus rastas morfologiniame žodyne. Įvedę minėtus pakeitimus, galėsime lengvai atpažinti trumpinius, vardus ir pavadinimus, ir jeigu šiems žodžiams nerasime atitikmens morfologiniame žodyne, įvesime papildomą analizę, tokių išraiškų kalbos dalims nustatyti.

Po minėtų žingsnių, jau turime tekstą suvestą į didelį masyvą, kuriame užfiksuotas tiek pats žodis, tiek jo eilės tvarka sakinyje ir paragrafe. Tačiau to negana. Dar viena svarbi detalė padedanti nagrinėti tekstą yra skyrybos ženklai - jie gali padėti atskirti sudėtinio sakinio dėmenis, kas stipriai pagelbsti sintaksinėje analizėje. Taigi, atskyrę sakinius į žodžius, šalia fiksuojame ir kokie skyrybos ženklai buvo panaudoti po to žodžio. Šią informaciją taip pat išsaugome į masyvą, drauge su jau turima informacija.

Imame pastebėti, kad vis daugiau informacijos reikia išsaugoti tame pačiame masyve, dėl to, patogumo dėlei, šiuos duomenis susiimportuojame į *Microsoft SQL Server* duombazę ir tolimesnę analizę atliksime ten. Taigi, masyvą pakeičiame į duomenų bazės lentelę, kurioje saugosime visą susijusią informaciją su kiekvienu žodžiu.

Iki šio žingsnio jau turime tekstą išskaidytą į atskirus žodžius. Dabar, norint imtis sintaksinės analizės, pirmiausia reikia nustatyti kiekvieno žodžio kalbos dalį, giminę, skaičių ir linksnį (jeigu tai aktualu kalbos daliai). Šiam tikslui pasiekti naudosimės lengvai prieinamu internetiniu kalbos dalių žodynu *www.morfologija.lt*. Paduodami žodį į žodyną, atgal gauname visas, žodį apibūdinančias, charakteristikas. Žodžio kalbos dalies paieška yra atliekama Python programoje - čia sugeneruojame užklausą kiekvienam teksto žodžiui pagal 5 paveiksliuke pateiktą šabloną. Pagal šią užklausą atgal gauname HTML kodo dalį, kurioje surašytos duoto žodžio charakteristikos. Svarbu paminėti, kad tas pats žodis gali turėti keletą reikšmių bei turėti keletą žodžio aprašymų, pavyzdžiui, "kovas" - arba daiktavardžio vyriškos giminės vienaskaita (mėnesio pavadinimas) arba daiktavardžio moteriškos giminės daugiskaita (nuo žodžio "kova"). Atlikę šiek tiek HTML kodo analizės, iš gauto atsakymo galime išskirti visas reikalingas dalis ir sudaryti nedidelį masyvą, kurio nariai yra tekstinės eilutės, kaip pateikta 13 paveiksliuke. Gavę šį masyvą, jį taip pat išsaugome duomenų bazėje.

```
1 uzklausa = 'http://www.morfologija.lt/zodzio-formos/' + zodis.lower()
```

5 išeities kodas. *www.morfologija.lt* žodžio užklausos šablonas

Apibendrindami šį poskyrį, galime išvardinti esmines problemas, su kuriomis susiduriame, skaidydami tekstą į atskiras dalis:

- Tokie trumpiniai, kaip vardo pirma raidė ir pavardė (J. Jablonskis), ne visuomet atpažįstami Python funkcijos ir dėl to sakiniai kartais yra netaisyklingai atskiriami. Šiai problemai spręsti reikia papildomos analizės prieš skaidant paragrafus į sakinius arba tobulesnio metodo sakinių atskyrimui, kuris būtų sukurtas remiantis lietuvių kalba.

- Trumpiniai su taškais ("A.R.S" arba "J. Jablonskis") turi būti traktuojami kaip vienas žodis, tačiau yra atskiriami dėl skyrybos ženklų, einančių su jais. Šiai problemai spręsti įvedėme žodžių pakeitimus, kurie pašalina skyrybos ženklus trumpiniuose.

- Tokie trumpiniai, kaip "128 mln.", "10 tūkst." ir panašiai, klaidingai nurodo sakinio pabaigą ir dėl to, sutinkant šiuos trumpinius, sakinys yra klaidingai skeliamas į du. Be to, šie trumpiniai nėra atpažįstami kaip tikri trumpiniai (nes rašomi mažosiomis raidėmis) ir jiems negalioja pakeitimai. Dėl to, įvedėme papildomą tokių trumpinių atpažinimą duomenų bazėje ir klaidingai atskirtus sakinius ten sujungiame iš naujo.

3.3. Sintaksinė analizė

Mūsų metode, sintaksinė sakinių analizė užima vieną pagrindinių vaidmenų. Analizė mums leis atpažinti pagrindines kalbos detales - veiksnį ir tarinį, bei detales kurios nėra esminės, o tik papildo informaciją. Čia į pagalbą pasitelksime D. Šveikauskienės daktaro disertacijoje [18] aprašytą metodą lietuviškų sakinių sintaksinei analizei. Šiame darbe buvo aprašyta didelė dalis lietuviškų sakinių gramatikos. Vienas iš pavyzdžių yra pateiktas 14 paveikslėlyje. Naudodamiesi šia gramatika galime nustatyti, kas yra sakinio veiksnys ir tarinys - išskirti svarbiausią sakinio dalį - kas ir ką veikė. Kitos dalys nėra tiek svarbios todėl papildomai jų neatskiriame.

Prieš tai buvusiame skyrelyje, gavome žodžių seką, su jiems priskiriamomis kalbos dalimis ir skyrybos ženklais. Reikia nepamiršti, kad kalbos dalis gali būti nusakyta nevienareikšmiškai. Be to, galime sutikti žodžių (vardų ir pavadinimų), kuriems nebūtinai rasime kalbos dalį - taip nutinka, kai užsienietiški vardai ir pavadinimai yra sulietuvinami. Dėl to reikia papildomo žingsnio, kad nustatytume arba patikslintume priskiriamą kalbos dalį.

Kalbos dalies patikslinimui skirti žingsniai:

1. Kalbos dalies nustatymas pagal galūnę. Vardai ir pavadinimai yra daiktavardžiai, dėl to nesunku nustatyti kai kuriuos linksnius vien tik pagal galūnę:

    - Jeigu žodis baigiasi raidėmis "ą", "ę", "į", vadinasi daiktavardis bus galininko linksnio vienaskaita. Deja, negalime patikslinti vyriška tai giminė ar moteriška, kadangi šios galūnės būdingos abiems giminėms.
    - Jeigu žodis baigiasi raide "ų", vadinasi daiktavardis bus kilmininko linksnio daugiskaita. Taip pat negalime patikslinti giminės.

2. Kalbos dalies patikslinimas, prielinksnių pagalba. Lietuvių kalboje, turime nemažą sąrašą įvairių prielinksnių (dalis jų pateikta 3 lentelėje), kurie visuomet eina prieš tam tikrą linksnį ([15]). Vadovaudamiesi šia taisykle, galime patikrinti tokius dalykus:

    - Kai kurie prielinksniai sutampa su daiktavardžių ar veiksmažodžių formomis (pavyzdžiui "greta (ko?)" ir "Greta", arba "ties (kuo?)" ir "ties kelią") arba yra naudojami kaip dalelytės ar jungtukai (pavyzdžiui "per"). Tikrindami, po galimo prielinksnio einantį žodį, galime nustatyti ar tai tikrai yra prielinksnis, ar kita kalbos dalis.

Prielinksnis	Linksnis	Prielinksnis	Linksnis	Prielinksnis	Linksnis
ant	Kilmininkas	apie	Galininkas	palei	Galininkas
be	Kilmininkas	skradžiai	Galininkas	paskum	Galininkas
prie	Kilmininkas	į	Galininkas	išilgai	Galininkas
dėl	Kilmininkas	per	Galininkas	po	Įnagininkas
iš	Kilmininkas	po	Galininkas	su	Įnagininkas
anot	Kilmininkas	prieš	Galininkas	sulig	Įnagininkas
greta	Kilmininkas	už	Galininkas	ties	Įnagininkas

3 lentelė. Prielinksnių poros su atitinkamais linksniais.

- Tikrindami prielinksnį, einantį prieš žodį, galime patikslinti to žodžio linksnį.

3. Kalbos dalies patikslinimas, įvardžių pagalba.

   - Įvardžiai "kuri", "kuris", "kurią" ir panašiai, dažniausiai jungia sudėtinius sakinius, dėl to šis įvardis yra nuoroda į kažkurį, prieš tai buvusį, daiktavardį (arba vardą/pavadinimą). Pagal skaičių, giminę ir/arba linksnį, galime patikslinti to daiktavardžio ypatybes.
   - Lygiai taip pat, galime patikslinti ir įvardžio ypatybes. Pavyzdžiui, įvardis "kurios", gali reikšti tiek daugiskaitos moteriškos giminės vardininką, tiek vienaskaitos moteriškos giminės kilmininką. Ieškodami šį įvardį atitinkančio daiktavardžio, galime nustatyti tikrąjį linksnį.

4. Kalbos dalies patikslinimas prieveiksmių pagalba. Prieveiksmiai, panašiai kaip prielinksniai, eina prieš tam tikras kalbos dalis. Tiksliau, eina prieš asmenuojamus veiksmažodžius arba prieš linksniuojamus dalyvius ir būdvardžius. Dėl to, sutikę prieveiksmį, tikimės po jo vienos iš minėtų dalių.

5. Kalbos dalies patikslinimas, ieškant derančių kalbos dalių. Pavyzdžiui, moteriškos giminės būdvardis dera su greta esančiu moteriškos giminės daiktavardžiu, būdvardžio daugiskaita, derės su daiktavardžio daugiskaita ir panašiai.

6. Kalbos dalies patikslinimas pagal iš eilės einančius veiksmažodžius. Įprastuose lietuviškuose sakiniuose du veiksmažodžiai eis iš eilės tik jei bus atskirti skyrybos ženklais, arba jeigu bent vienas iš tų veiksmažodžių bus bendratis. Kitu atveju, vienas iš tų žodžių greičiausiai nebus veiksmažodis. Naudodamiesi šia prielaida, galime pašalinti netikrus veiksmažodžius.

7. Kalbos dalies patikslinimas, pagal vyraujančias galūnes. Surinkdami visas tekste esančias galūnes, drauge su priklausančiomis kalbos dalimis ir ypatybėmis, galime nustatyti vardams ir pavadinimams (kurie nebuvo nustayti prieš tai) priklausančias kalbos dalis. Tokių galūnių pavyzdys pateiktas 4 lentelėje.

Visi aukščiau išvardinti punktai, mums leidžia patikslinti žodžio kalbos dalį, arba bent to žodžio linksnį, giminę ar skaičių. Atlikę šiuos patikslinimus nebūtinai gausime unikalias kalbos dalis, tačiau tai mums leis lengviau ir tiksliau surasti sakinio veiksnį ir tarinį. Kaip žinome iš lietuvių kalbos gramatikos, veiksnys ir tarinys yra pagrindinės sakinio dalys, nusakančios sakinio veikėją ir jo atliekamus veiksmus. Turėdami šias sakinio dalis, galėsime maždaug numanyti, apie ką yra

Galūnė	Linksnis	Giminė	Skaičius	Kalbos dalis
ba	Įn.	mot.g.	vns.	Daiktavardis
ba	Š.	mot.g.	vns.	Daiktavardis
ba	V.	mot.g.	vns.	Daiktavardis
bai	N.	mot.g.	vns.	Būdvardis
bas	G.	mot.g.	dgs.	Daiktavardis
bą	G.	mot.g.	vns.	Daiktavardis

4 lentelė. Galūnės ir jų ypatybės, sugeneruotos bandymų metu.

pasakojama sakinyje ar paragrafe.

Kaip teigiama D. Šveikauskienės daktaro disertacijos darbe ([18]), būdingiausia veiksnio forma yra vardininko linksnis. Daug rečiau veiksniu tampa kilmininko linksnis arba veiksmažodžio bendratis. Lituanistai pateikia net penkias kalbos dalis, kurios galėtų būti veiksniu, tai - daiktavardis, įvardis, skaitvardis, būdvardis ir dalyvis. Toliau savo darbe naudosimės D. Šveikauskienės sudarytu veiksnio aprašu, kuris pavaizduotas 14 paveikslėlyje. Tiesa, ji į veiksnio aprašą įtraukė ne visus įmanomus veiksnio variantus, o tik dažniausiai sutinkamus. Taigi, analogiškai pasielgsime ir mes.

Aprašyti tarinį yra gerokai sudėtingiau, kadangi jis gali turėti daug išreiškimo būdų ir formų. Tariniai lietuvių kalboje gali būti vientisiniai arba sudėtiniai. Šios tarinių grupės atitinkamai dar skirstomos į asmenuojamus - neasmenuojamus ir suvestinius, sudurtinius ir mišriuosius. [18] šaltinyje D. Šveikauskienė aprašo didžiąją dalį visų šių tarinių, praleisdama rečiau vartojamus tarinio variantus. Kadangi mūsų darbo tikslas nėra pilna sintaksinė sakinių analizė, darome prielaidą, kad nebūtina pilnai atpažinti visų tarinių ir ieškosime tik dažniausiai sutinkamų tarinio variantų. Taigi, pasiremdami [7] šaltiniu, žemiau pateikiame visus įmanomus tarinio variantus:

1. Vientisinis (grynasis) tarinys - išreikštas vientisine veiksmažodžio forma. Šis tarinys su veiksniu derinamas asmeniu ir skaičiumi. Gali būti išreikštas šiomis kalbos dalimis:
    - Asmenuojamomis formomis (seku, ėjome, žaisdavome, sėdėk ir t.t.);
    - Dalyviu (einąs, eidavęs, buvęs ir t.t.);
    - Bendratimi (eiti, vyti ir t.t.) - šios formos neįtrauksime dėl reto naudojimo;
    - Ištiktuku (taukšt, šmakšt ir t.t.) - šios formos neįtrauksime dėl reto naudojimo.
2. Suvestinis tarinys - tarinys, išreikštas veiksmažodžio vientisine forma kartu su bendratimi:
    - Pirmąją šio tarinio dalį sudaro veiksmažodžių asmenuojamos formos - būti, pradėti, baigti, nustoti, imti, reikėti, turėti, galėti, norėti, drįsti, mėginti, bandyti, ketinti, ruoštis ir kitos;
    - Antrąją dalį sudaro veiksmažodžio bendratis;
    - Pavyzdžiui - pradėjo matyti, ėmė suprasti ir t.t.
3. Sudurtinis tarinys - tarinys, išreikštas veiksmažodžio asmenuojamąja forma ir linksniuojamuoju žodžiu:
    - Pirmoji tarinio dalis išreiškiama veiksmažodžių - būti, likti, tapti, pasidaryti, atrodyti, apsimesti, jaustis, virsti asmenuojamomis formomis;

<VEIKSNYS> ::= <VEIKSN-DAIKT> | <VEIKSN-ĮVARD-*DAIKT*> | <VEIKSN-BENDRAT>;

<VEIKSN-DAIKT> ::= <VEIKSN-DAIKT-VARD-VNS-VYRG> |
<VEIKSN-DAIKT-VARD-VNS-MOTG> |
<VEIKSN-DAIKT-VARD-DGS-VYRG> |
<VEIKSN-DAIKT-VARD-DGS-MOTG>;

<VEIKSN-ĮVARD-*DAIKT*> ::= <VEIKSN-ĮVARD-VARD-VNS-VYRG-*DAIKT*> |
<VEIKSN-ĮVARD-VARD-VNS-MOTG-*DAIKT*> |
<VEIKSN-ĮVARD-VARD-DGS-VYRG-*DAIKT*> |
<VEIKSN-ĮVARD-VARD-DGS-MOTG-*DAIKT*> |
<VEIKSN-ĮVARD-BEVG>;

<VEIKSN-BENDRAT> ::= <VEIKSN-BENDR> |
<VEIKSN-BENDR-*KILM*> |
<VEIKSN-BENDR-*NAUD*> |
<VEIKSN-BENDR-*GAL*> |
<VEIKSN-BENDR-*ĮNAG*> |
<VEIKSN-BENDR-*VIET*>;

14 pav. D. Šveikauskienės aprašytos gramatikos ištrauka.

- Antroji dalis gali būti išreiškiama:
  - Daiktavardžio, būdvardžio, skaitvardžio, įvardžio ir būdvardiškai vartojamų dalyvių vardininku;
  - Daiktavardžio kilmininku;
  - Daiktavardžio įnagininku;
  - Linksniais su prielinksniais.
- Pavyzdžiui - buvo graži, liko vienas ir t.t.

4. Mišrusis tarinys - turi ir suvestinio, ir sudurtinio tarinio savybių:
   - Pavyzdžiui - nori būti pirmas, ėmė ruoštis namo ir t.t.

Turėdami veiksnio ir tarinio aprašymą, galėtume rasti kiekvienam teksto sakiniui jo centrą. Tačiau, minėtas aprašas veiks tik vientisiniams sakiniams, kai tuo tarpu mūsų tekste galima sutikti ir sudėtingų sudėtinių sakinių. Dėl to, reikia dar vieno papildomo žingsnio - sudėtinių sakinių išskaidymo.

Sudėtiniai sakiniai, tai sakiniai turintys du ir daugiau gramatinių centrų (veiksnys + tarinys). Galime išskirti tris pagrindines tokių sakinių grupes:

1. Sujungiamuosius jungtukinius - kuriuose dėmenys jungiami sujungiamaisiais jungtukais (ir, ar, arba, o, bet, tačiau, vis dėlto, tik(tai), užtai, todėl, tad, dėl to, tai, taigi);

2. Prijungiamuosius jungtukinius - kuriuose dėmenys jungiami prijungiamaisiais jungtukais (kad, jog, ne, kadangi, jeigu, jei), santykiniais įvardžiais (kas, koks, kuris, kelintas), prieveiksmiais (kada, kol, kaip, kiek, kur, kame), dalelytėmis (ar, vos tik, tarsi, tartum);

3. Bejungtukius - kuriuose dėmenys jungiami tik intonacija ir prasme;

Lengviausiai atpažįstami yra pirmo ir antro tipo sudėtiniai sakiniai, kadangi juos nurodo jungtukas ir skyrybos ženklas, kurie (beveik) visuomet eina kartu, kai tuo tarpu bejungtukiai sudėtiniai sakiniai tekste bus atskiriami tik skyrybos ženklais. Kaip žinome, skyrybos ženklai nebūtinai skiria

du sakinius, bet taip pat gali skirti ir dvi aplinkybes, du veiksnius ar kitas sakinio dalis. Dėl šios priežasties trečiojo tipo sakinių neskirsime ir tikėsimės, kad tokių sakinių bus nedaug, kadangi jie yra būdingi vaizdingai kalbai, kuri dažniausiai naudojama grožinėje literatūroje ir labai retai sutinkama straipsniuose.

Apibendrinant šį poskyrį, turėdami teksto žodžių masyvą (lentelę duomenų bazėje), turime kiekvienam žodžiui gauti jo kalbos dalį (arba dalis). Tuomet pagal šią kalbos dalį ir skyrybos ženklus galime atpažinti sudėtinius sakinius tekste bei juos atskirti į savarankiškus sakinius. Galiausiai, kiekvienam iš šių naujai suformuotų sakinių galime rasti sakinio centrą, pasiremdami šiame skyrelyje sudarytu veiksnio ir tarinio aprašu. Turint omenyje, kad kai kurie žodžiai gali atitikti kelias skirtingas kalbos dalis, gali būti, kad ir veiksnys ar tarinys bus nustatytas nevienareikšmiškai. Tokiu atveju darome patikrinimus veiksnio ir tarinio poroms:

1. Kiekvienas veiksnys turi turėti derantį tarinį, todėl, jeigu nerandame jam derančio tarinio, pasirinktas veiksnys nėra tikras veiksnys;

2. Kiekvienas tarinys turi turėti derantį veiksnį, todėl jeigu nerandame jam derančio veiksnio, pasirinktas tarinys nėra tikras tarinys;

Žemiau išvardiname pagrindines problemas, su kuriomis susidūrėme atlikdami sintaksinę analizę ir punktus, į kuriuos reikėtų atkreipti dėmesį:

1. Pavadinimai ir vardai, pavardės nėra įtraukti į *www.morfologija.lt* žodyną, todėl negalime rasti jų kalbos dalies. Tačiau, vardai ir pavadinimai labai dažnai gali būti veiksniu ir tai sukelia riziką, kad veiksnys bus neatpažintas sakinyje arba bus atpažintas neteisingai. Šiai problemai spręsti įvedėme keletą kalbos dalies atspėjimo žingsnių:

    - Įvedėme kalbos dalies nustatymą, remdamiesi keliomis labai specifinėmis daiktavardžių galūnėmis;
    - Įvedėme kalbos dalies ypatybių nustatymą, pagal vyraujančias galūnes tekste;
    - Įvedėme kalbos dalies ypatybių nustatymą, pagal prielinksnius;

2. Be vardų ir pavadinimų, yra ir kitų žodžių, kurie nėra įtraukti į žodyną. Tai nebūtinai žodžiai, kuriuos kažkas sieja, bet gali būti atsitiktiniai žodžiai. Šiai problemai spręsti įvedėme papildomus žingsnius, paminėtus anksčiau šiame poskyryje.

3. Nėra aišku, kaip nustatyti žodžio kalbos dalį (ar tas žodis gali būti pavadinimas?), kai žodis yra pirmasis sakinio žodis ir rašomas didžiąja raide, tačiau nėra randamas žodyne. Tokiems žodžiams galima bandyti atspėti kalbos dalį su ypatybėmis, pagal vyraujančias galūnes tekste, tačiau toks būdas gali grąžinti daug skirtingų variantų.

4. Tikslesnei sintaksinei analizei reikalingas tikslesnis ir išsamesnis morfologinis žodynas.

5. Tikslesnei sintaksinei analizei reikalingas efektyvesnis metodas, atskiriantis sudėtinių sakinių dėmenis.

3.4. Konteksto išskyrimas

Žodis "kontekstas" yra kilęs iš lotyniško žodžio *"contextus"*, kuris sudarytas iš dviejų dalių: *"con"* - kartu ir *"texere"* - susieti, supinti. Taigi, kontekstas turėtų būti mintis, susiejanti žodžius,

sakinius ir paragrafus.

Pagrindinė detalė, siejanti teksto dalis į bendrą visumą yra teksto veikėjai. Darome prielaidą, kad turėdami pagrindinius veikėjus ir žinodami, ką jie veikė, turėsime ir pagrindinę viso teksto mintį. Taigi, pasinaudodami 3.3 poskyryje aprašytais veiksniu ir tariniu, randame kiekvieno sakinio centrą. Kaip pavaizduota 6 kodo ištraukoje, 1-oje eilutėje sugeneruojame sąrašą visų sakinių su jų veiksniais, 6-oje eilutėje sugeneruojame sąrašą sakinių su jų tariniais, o 11-oje eilutėje šiuos du sąrašus sujungiame, kad gautume pilną sakinio centrą. Čia *dbo.SplitText* žymi duomenų bazės lentelę, kurioje yra visi teksto žodžiai, sunumeruoti pagal paragrafo, sakinio ir žodžio eilės tvarką. Šioje lentelėje turime visus skyrybos ženklus, bei visas žodžio kalbos dalis. Simboliai v^* ir t^* atitinkamai žymi visus įmanomus veiksnio ir tarinio išraiškos būdus.

Pagrindinis veikėjas turėtų būti tas, kuris yra dažniausiai minimas visame tekste, ne tik kaip sakinio veikėjas, bet ir apskritai visose kitose sakinio dalyse. Atpažinti pagrindinį veikėją mums padės statistinė analizė - randame visus žodžius, pasikartojančius tekste dažniausiai. Bet prieš tai, iš visos turimų žodžių aibės turime atmesti žemiau išvardintus:

- Triukšmo žodžius - žodžiai, kurie dažnai pasikartoja, bet neneša informacijos. Sąrašą tokių žodžių galima rasti [8] šaltinyje;
- Prieveiksmius - šie žodžiai dažniausiai sustiprina įspūdį (labai, baisiai ir pan.), tačiau neturi kontekstinės informacijos;
- Dalelytes, ištiktukus, jaustukus - šie žodžiai dažniausiai naudojami pagražinti tekstą, tačiau taip pat neturi kontekstinės informacijos;
- Įterpinius - žodžius ir žodžių junginius, kuriais pasakomas požiūris į dalyką ar pateikiama su sakinio turiniu susijusių pastabų. Tokie įterpiniai gali būti išskiriami kableliais, brūkšniais ar skliaustais.

Taigi, imame visus teksto žodžius (unigramas) ir juos sugrupuojame pagal žodžio lemą ir skaičiuojame kiekvieno žodžio pasikartojimus (kodas nuo 17 eilutės). Atmetame visus žodžius, kurie tekste pasikartojo tik vieną kartą ir iš likusios dalies žodžių randame dažniausius pagal 2.17 formulę. Rasti dažniausiai pasikartojančius žodžius taip pat galima ir užsibrėžiant ribas, tarp kurių atsirandantys žodžiai vadinami reikšmingiausiais, kaip aprašyta 2.3.1 skyrelyje ir pavaizduota 6 paveikslėlyje.

Toliau, ieškosime veiksnių ir dažniausių žodžių aibių sankirtos - tokiu būdu rasime pagrindinius teksto veikėjus. Tą galima matyti nuo 25 kodo eilutės - pirmiausia jungiame dažniausių žodžių aibę su veiksnių aibe, kad gautume pagrindinius veikėjus. Tuomet prie šių veikėjų prijugiame jiems priklausančius tarinius iš sakinių centrų aibės. Tokiu būdu sugeneruojame frazes, sakinius ir teiginius, kurie bus visos santraukos pagrindas.

```
1 SELECT SentenceID, Word
2 INTO #veiksniuAibe
3 FROM dbo.SplitText
4 WHERE Grammar IN (v*);
5
6 SELECT SentenceID, Word
7 INTO #tariniuAibe
8 FROM dbo.SplitText
9 WHERE Grammar IN (t*);
10
```

```
11 SELECT VA.SentenceID, VA.Word AS Veiksnys, TA.Word AS Tarinys
12 INTO #centruAibe
13 FROM #veiksniuAibe AS VA
14 INNER JOIN #tariniuAibe AS TA
15                     ON TA.SentenceID = VA.SentenceID;
16
17 SELECT LEMMA(Word) AS Word, COUNT(*) AS WordCount
18 INTO #dazniausiuZodziuAibe
19 FROM dbo.SplitText
20 WHERE Word NOT IN (StopList)
21 GROUP BY LEMMA(Word)
22 HAVING COUNT(*) > 1
23 AND COUNT(*) > MEDIAN;
24
25 SELECT CA.Veiksnys, CA.Tarinys
26 FROM #dazniausiuZodziuAibe AS DZA
27 INNER JOIN #veiksniuAibe AS VA
28                     ON LEMMA(VA.Word) = LEMMA(DZA.Word)
29 INNER JOIN #centruAibe AS CA
30                     ON LEMMA(CA.Veiksnys) = LEMMA(VA.Word);
```

6 išeities kodas. Konteksto išskyrimo pseudokodas

3.5. Santraukos generavimas

Buvusiame poskyryje aprašėme, kaip galime gauti pagrindinius teksto veikėjus su jiems priklausančiais sakinių centrais. Pagal šiuos žodžių junginius galėtume rasti originalius sakinius ir juos pateikti skaitytojui, išrikiuotus eilės tvarka. Tokiu atveju gautume ištraukomis paremtą santrauką, kurios skaitomumas bus pakankamai neblogas, tačiau sakiniai tarpusavyje nebūtinai jungsis, gali atsirasti minties šuolių. Be to, sakiniai gali būti nepagrįstai ilgi ir turėti perteklinių sakinio dalių.

Kaip minėjome pradžioje, mūsų tikslas yra gauti santrauką panašesnę į tokią, kurią būtų parašęs žmogus. Dėl to prieš grąžindami jau sugeneruotas frazes, jas dar šiek tiek pataisome. Čia galime iš mūsų gautų frazių sugeneruoti 1, 2, 3 ar 4 laipsnio n-gramas (priklausomai nuo gautų frazių ilgio) ir jas palyginti su n-gramų žodynu - taip galėtume patikrinti ar frazė yra taisyklinga ir ar apskritai naudojama. Netaisyklingoms frazėms galėtume rasti jų atitikmenis tame pačiame žodyne pasinaudoję kokia nors panašumo funkcija. Be to, pasinaudodami mikro-nuomonių metodu, aprašytu 2.4.1 skyrelyje, galime sujungti vieno veiksnio panašius tarinius į vieną frazę. Tokiu būdu dar labiau koncentruojame informaciją bei mažiname santraukos ilgį.

3.6. Testavimas

Suvedus tekstą į santrauką, reikia įvertinti du dalykus - kaip tiksliai gautas teksto trumpinys atspindi viso teksto esmę ir gautos santraukos skaitomumą.

Patikimiausias būdas santraukos metodui testuoti yra žmogiškasis testavimas. Vienas didžiausių trūkumų čia yra testuotojo subjektyvumas - kas vienam atrodys gera santrauka, kitam gali atrodyti visai priešingai. Norint išvengti tokio subjektyvumo, reikėtų bent keleto testuotojų, kurie vertintų santraukos kokybę ir vėliau galėtume jų pateiktus rezultatus suvidurkinti. Kitas labai svarbu faktorius - toks testavimo metodas yra labai imlus laikui - reikia perskaityti ir įsigilinti tiek į

visą tekstą, tiek į pateiktą santrauką. Dėl šios priežasties ieškome patogesnių ir greitesnių metodų kokybei patikrinti.

Norint automatizuoti metodo santraukos kokybės testavimą, pirmiausia reikia aibės tekstų su jų santraukomis, kad turėtume su kuo palyginti. Turint tokią aibę, galima gautą santrauką palyginti su jau turima to paties teksto santrauka. Jeigu pirma santrauka yra lygi antrajai - metodas veikia 100 procentų, tačiau labai maža tikimybė, kad taip ir bus. Dėl to reikia tikrinti, kaip arti gauta santrauka yra prie turimos. Tą apskaičiuoti galima naudojant *Python* kalbos biblioteką *difflib*. Ši biblioteka turi funkciją *SequenceMatcher*, kuri nustato dviejų tekstų panašumą ir grąžina skaičių nuo 0 iki 1, kur 0 reiškia, kad du tekstai visiškai skiriasi, o 1 - tekstai visiškai sutampa. Ši funkcija nuosekliai ieško ilgiausio kaimyninių žodžių junginio, kuris pasikartotų tarp dviejų lyginamų frazių, dėl to yra įvertinama net ir žodžių tvarka bei didžiosios - mažosios raidės. Vadinasi, naudojantis funkcija, du sakiniai - "spintoje kabojo gėlėta suknelė" ir "gėlėta suknelė kabojo spintoje" bus panašūs vos 46,66%. Dėl to, šią funkciją reikėtų taikyti kiekvienam santraukos sakiniui ir jį lyginti su kiekvienu kitu turimos santraukos sakiniu, papildomai dar konvertuojant visas raides vien į mažąsias arba didžiąsias. Jeigu tarsime, kad $S^*(s_1, ..., s_k)$ žymi gautą santrauką, o $\bar{S}(\bar{s}_1, ..., \bar{s}_m)$ turimą teksto santrauką, tai S^* ir $\bar{S}$ panašumą galime įvertinti taip:

$$SIM(S^*, \bar{S}) = \frac{1}{k} \times \max_{\forall s_i \in S^*} (SequenceMatcher(s_i, \bar{s}_m),\ \forall \bar{s}_m \in \bar{S}) \tag{3.1}$$

Tuomet šios dvi santraukos bus pakankamai panašios, jeigu $SIM(S^*, \bar{S}) >= 0.5$.

Ką tik minėta funkcija patikrins, kaip stipriai du tekstai yra panašūs, tačiau nepasakys, ar tekstas yra taisyklingas ir ar yra pakankamai skaitomas. Dėl to, reikia dar vieno metodo, kuris būtent skaitomumą ir išmatuotų. Čia galime pasinaudoti [13] šaltinyje aprašyto metodo skaitomumo įverčio formule, pavaizduota 3.2 lygybėje. Čia S_{read} žymi skaitomumo įvertį nuo frazės ar sakinio $(w_1, ..., w_n)$, K yra visų apskaičiuotų sąlyginių tikimybių skaičius, q - pasirinktas n-gramų lapsnis. Praktiškai, ši formulė reiškia, kad turimą sakinį skaidome į pasirinkto laipsnio n-gramas, kurioms skaičiuojame tikimybes, kad žodis w_k eis po žodžių $w_{k-q+1}...w_{k-1}$. Šias sąlygines tikimybes suvidurkiname, padalindami iš viso jų skaičiaus K. Sąlyginė tikimybė $P(w_k|w_{k-q+1}...w_{k-1})$ bus apskaičiuojama, kaip n-gramos $w_{k-q+1}...w_{k-1}w_k$ pasikartojimų skaičius padalintas iš visų n-gramų žodyne.

$$S_{read}(w_1, ..., w_n) = \frac{1}{K} \times \log_2 \prod_{k=q}^{n} P(w_k|w_{k-q+1}...w_{k-1}) \tag{3.2}$$

3.7. Nuveiktų darbų apibendrinimas

Dauguma automatinių santraukos metodų yra paremti ištraukomis iš teksto - paprastai yra išimamas tam tikras skaičius sakinių iš originalo, kurie stipriausiai atspindi teksto esmę, ir pateikiama skaitytojui. Tačiau, tokiu būdu gautos santraukos nebūna rišlios, o gauta sakinių seka labiau primena išvardintus teiginius negu bendrą tekstą. Dėl to, mūsų pagrindinė idėja šiame darbe buvo sukurti tokį metodą, kuris generuotą santraukas, panašias į parašytas žmogaus.

Šiam tikslui pasiekti įvedėme sintaksinę sakinių analizę - mūsų nagrinėtuose kitų autorių darbuose sintaksinė analizė nebuvo naudojama. Kėlėme hipotezę, kad nagrinėdami tekstą iš gramatinės pusės gausime tikslesnę santrauką, bei gautas tekstas bus rišlesnis ir trumpesnis. Čia susidūrėme su problema, kad iki šiol neturime pilnos lietuviškų sakinių analizės (tai yra atlikta tik vientisiniams sakiniams) ir dėl to mūsų atlikta sintaksinė analizė taip pat nėra labai tiksli.

Taip pat, sukūrėme teorinį modelį, kaip tekstas gali būti suvedamas į santrauką, pasinaudojant

sintaksine ir statistine analize. Šis metodas kol kas nebuvo pilnai įgyvendintas programatiškai, dėl to negalime pateikti jo efektyvumo įvertinimo. Tačiau, pateikiame nuorodas, kaip įgyvendintą metodą galima ištestuoti ir įvertinti.

Išvados ir rekomendacijos

- Prieš imantis automatinės santraukos generacijos, reikėtų gerai apsvarstyti, kam bus naudojama ši santrauka ir kokia ji turėtų būti. Jeigu norime santrauką sugeneruoti oficialiam tekstui, ar pateikti knygoje, tuomet mums reikės taisyklingos ir rišlios santraukos. Jeigu norime tiesiog supažindinti su tekstu ir maždaug aprašyti, apie ką jis bus, tuomet mums užteks keleto sakinių išimtų iš to teksto. Pirmuoju atveju, teksto analizė reikalauja labai daug resursų ir turi sudėtingą implementaciją. Be to, tokį metodą bus sunku pritaikyti kitiems tekstams - dėl šių priežasčių, dažnai pasirenkamas antrasis variantas. Antruoju atveju galime pritaikyti daug įvairių statistinių metodų ir tikimybiškai rasti tuos sakinius, kurie geriausiai atspindi turinį, nesiveldami į sudėtingą sintaksinę analizę. Kurti apibendrinimais paremtą santrauką dažnai neapsimoka, kadangi naudos gauname mažiau, negu įdedame savo darbo ir skiriame kitų resursų.

- Prieš imantis sudėtingesnių natūralios kalbos apdorojimo darbų, pirmiausia reikėtų turėti pilnai aprašytą lietuvių kalbos gramatiką. Naudojant tik dalinę jos versiją susiduriame su sunkumais atlikdami sintaksinę analizę.

- Lietuvių kalbos apdirbimui pasitarnautų kokybiškas metodas, atskiriantis paragrafą į atskirus sakinius, kadangi tai yra vienas iš dažniausiai sutinkamų žingsnių natūralios kalbos apdorojime.

- Lietuvių kalbos apdirbimui pasitarnautų nemokamas internetinis n-gramų žodynas su įverčiais, kuriame galėtume pasitikslinti frazių naudojamumą ir taisyklingumą.

Gairės

1. Patobulinti sintaksinės analizės dalį:
   (a) Įvesti geresnį metodą paragrafų atskyrimui į sakinius;
   (b) Įvesti išsamesnį žodyną, kuriame būtų randamos visos žodžių formos;
   (c) Patobulinti sudėtinių sakinių atpažinimą;
   (d) Įvesti platesnį veiksnio ir tarinio aprašymą;
2. Įvesti daugiau rodiklių, kurie nurodo turinio kontekstą, tikslesnei santraukai generuoti;
3. Užbaigti programinę metodo implementaciją.

Literatūros šaltiniai

[1] https://machinelearningmastery.com/natural-language-processing/.

[2] https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm.

[3] http://nlpforhackers.io/textrank-text-summarization/.

[4] https://www.theverge.com/2017/5/14/15637588/salesforce-algorithm-automatically-summarizes-text-

[5] http://www.yourdictionary.com/n-gram.

[6] http://www.dictionary.com/browse/n-gram.

[7] http://lietuviu7-8.mkp.emokykla.lt/lt/mo/zinynas/tarinio_rusys_ir_reiskimo_budai/.

[8] https://gist.github.com/revelt/01524e76c6e5e0970d2d0fe8797e92ed.

[9] Atviro kodo python programa žodžių lemavimui. https://bitbucket.org/sirex/morfologija/overview.

[10] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009. http://www.nltk.org/book/.

[11] Dipanjan Das and André FT Martins. A survey on automatic text summarization. *Literature Survey for the Language and Statistics II course at CMU*, 4:192–195, 2007.

[12] Vidas Daudaravičius, Erika Rimkutė, and Andrius Utka. Morphological annotation of the lithuanian corpus. Balto-Slavonic Natural Language Processing 2007, Pages 94-99, June 29 2007.

[13] Kavita Ganesan, ChengXiang Zhai, and Evelyne Viegas. Micropinion generation: an unsupervised approach to generating ultra-concise summaries of opinions. In *Proceedings of the 21st international conference on World Wide Web*, pages 869–878. ACM, 2012.

[14] Petras Kniūkšta. Kompiuterinis lietuvių kalbos žinynas. nuo morfologijos iki reikalų raštų. SKYRYBA, 2004. http://www.xn--altiniai-4wb.info/files/kalba/KL00/Lietuvi%C5%B3_kalbos_%C5%BEinynas._Skyryba.KL0001.pdf.

[15] Petras Kniūkšta. Kompiuterinis lietuvių kalbos žinynas. nuo morfologijos iki reikalų raštų. PRIELINKSNIŲ VARTOJIMAS IR REIKŠMĖ, 2004. http://www.xn--altiniai-4wb.info/files/kalba/KJ00/Lietuvi%C5%B3_kalbos_%C5%BEinynas._Prielinksni%C5%B3_vartojimas_ir_reik%C5%A1m%C4%97.KJ1806.pdf.

[16] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958. http://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf.

[17] Allan Ramsay. Grammars and parsing, 2000.
http://ccl.pku.edu.cn/doubtfire/NLP/Parsing/Introduction/Grammars%20and%20Parsing.htm.

[18] D Šveikauskienė. Lietuvių kalbos vientisinių sakinių automatinė sintaksinė analizė: daktaro disertacija, 2009.
http://old.mii.lt/files/mii_dis_san_2010_sveikauskiene.pdf.

[19] Vytautas Zinkevičius. Lemuoklis - morfologinei analizei. Darbai ir dienos, p. 245-273, 2000.
http://donelaitis.vdu.lt/publikacijos/zinkevicius.pdf.