LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

**DOKUMENTU MEKLĒŠANAS SISTĒMA**

KURSA DARBS

Autors: **Arnolds Sarmulis**

Studenta apliecības Nr.: as18284

Darba vadītāja: Dr.sc.comp. Lelde Lāce

**ANOTĀCIJA**

Kursa darbā tiek apskatīta dokumentu meklēšanas problēma digitālā vidē. Šī problēma tiek risināta izmantojot programmu, kas nodrošina automatizētu dokumentu meklēšanu pēc to satura vai citiem raksturlielumiem. Uzsvars ir pētīt kādas datu struktūras un kādi algoritmi var būt programmas pamatā kura nodrošina šīs problēmas risināšanu. Tiek apskatīti arī veidi kā lietotājs var veikt dokumentu meklēšanu programmā izmantojot lietotāja saskarni.

Problēmas fokuss ir uz dokumentu meklēšanu lokālā datu nesējā, vai serverī, dokumentu meklēšana internetā šajā darbā netiek apskatīta. Šī problēma ir aktuāla situācijās kur cilvēkam digitālā datu krātuvē vajag ātri atrast dokumentu ar noteiktu saturu, bet kādu iemeslu dēļ nav zināms kur tieši šis dokuments atrodas un kā to sauc. Manuāla meklēšana šādā situācijā var prasīt ļoti daudz laika, ko varētu izmantot lietderīgāk. Tad lietotājs izmantojot programmu veic dokumenta meklēšanu.

Atslēgvārdi: dokumentu meklēšana, meklēšanas programma, datu struktūras, algoritmi.

**ABSTRACT**

**Document seach system**

The problem of document search in the digital environment is considered in the course work. This issue is addressed through a program that provides automated search for documents by their content or other characteristics. The emphasis is on researching what data structures and algorithms can be the basis of a program that provides a solution to this problem. It also looks at ways a user can search for documents in the program using the user interface.

The focus of the problem is on searching for documents on a local data carrier or server, searching for documents on the Internet is not considered in this work. This problem is relevant in situations where a person needs to quickly find a document with certain content in a digital repository, but for some reason it is not known exactly where the document is located and what it is called. Searching manually in this situation can be very time consuming. The user then searches for a document using the program.

Keywords: document search, search engine, data structures, algorithms.

**SATURA RĀDĪTĀJS**

[1. Problēmas raksturojums 7](#_Toc93271402)

[1.1 Problēmas sastāvs 7](#_Toc93271403)

[1.2 Meklēšanas procesa apraksts 7](#_Toc93271404)

[1.3 Piemēri 8](#_Toc93271408)

[2. Dokumentu indeksēšana 9](#_Toc93271409)

[2.1 Ievads 9](#_Toc93271411)

[2.2 Bag of words 9](#_Toc93271412)

[2.2.1 Struktūras apraksts 9](#_Toc93271413)

[2.2.2 Struktūras izveidošana 10](#_Toc93271414)

[2.2.3 Realizācija divdimensionālā matricā 10](#_Toc93271415)

[2.2.4 Alternatīva realizācija divdimensionālā matricā 12](#_Toc93271416)

[2.2.5 Absolūtās vērtības realizācija 13](#_Toc93271417)

[2.2.6 Bag of words nobeiguma secinājumi 15](#_Toc93271418)

[2.3 Inverted Index 15](#_Toc93271419)

[2.3.1 Pamatvarianta struktūras apraksts 15](#_Toc93271420)

[2.3.2 Struktūras izveidošana 16](#_Toc93271421)

[2.3.3 Inverted Index ar metadatiem 18](#_Toc93271422)

[2.3.4 Struktūras izveidošana 18](#_Toc93271423)

[2.4 Struktūras uzturēšana 20](#_Toc93271424)

[2.5 Kopsavilkums par dokumentu indeksēšanu 21](#_Toc93271425)

[3. Dokumentu meklēšana un atgriešana 24](#_Toc93271426)

[3.1 Kopīgais piemērs 24](#_Toc93271428)

[3.2 Meklēšanas algoritmi 24](#_Toc93271429)

[3.2.1 Levenšteina algoritms 24](#_Toc93271430)

[3.2.2 Binārā meklēšana un inverted index 26](#_Toc93271431)

[3.2.3 Binārās meklēšanas algoritms 26](#_Toc93271438)

[3.2.4 Inverted index meklēšanas algoritms 27](#_Toc93271439)

[3.3 Vērtību iekļaušanas modelis 30](#_Toc93271440)

[3.4 Patiesumvērtību meklēšanas modelis 32](#_Toc93271441)

[3.4.1 Pamatvariants 33](#_Toc93271442)

[3.4.2 Paplašinātais variants 34](#_Toc93271443)

**APZĪMĒJUMU SARAKSTS**

**IEVADS**

Datorā veicot dažādus darbus var gadīties situācijas kurās ir nepieciešams ātri atrast kādu informāciju lokālā datņu krātuvē, bet precīzi nav zināms kur tā atrodas un pēc dokumenta nosaukuma ne vienmēr var uzminēt vai šis dokuments saturēs nepieciešamo informāciju. Mūsdienās cilvēki kuri strādā vai mācās ar vien biežāk lokālajā datora atmiņā mēdz glabāt lielus apjomus ar dažādiem dokumentiem, līdz ar to pārvaldīšana paņem aizvien vairāk laika. Meklējot nepieciešamo informāciju lietotājām nākas staigāt turpu šurpu pa dažādām mapēm, atvērt un lasīt datnes, kas ir diezgan laikietilpīgs process. Tātad manuāla informācijas meklēšana var paņemt daudz laika, ko varētu ieekonomēt ja šo darbību ir iespējams efektīvi automatizēt.

Darbā fokuss ir pētīt problēmas risinājumu iespējas atbilstoši situācijām kur cilvēkam kurš strādā vai mācās, vai kādu citu iemeslu dēļ datora atmiņā ir daudz un dažādi dokumenti, un ir vajadzība efektīvi un ātri atrast noteiktus dokumentus, kas satur nepieciešamo informāciju. Lai saprastu kā priekš lietotāja varētu automatizēti atrast viņam nepieciešamo informāciju ir nepieciešams pētīt meklēšanu digitālā vidē.

Darba mērķis ir iedziļināties un izpētīt dažādus veidus, kā var realizēt efektīvu meklēšanu. Izpētīt meklēšanai nepieciešamos algoritmus un datu struktūras un saprast kuri no tiem ir labāk piemēroti problēmas risināšanai.

Tiek izvirzīti sekojoši darbā veicamie uzdevumi:

1. Raksturot darbā risināto problēmu;
2. Izpētīt un salīdzināt datu struktūras, kas var būt dokumentu meklēšanas programmas pamatā;
3. Aprakstīt un novērtēt algoritmus, kas var tikt izmantoti programmā;
4. Sniegt dažādus saskarnes modeļus kā lietotājs var meklēt dokumentus izmantojot programmu.

Darba izstrādē tiek izmantotas sekojošas metodes:

1. Aprakstošā metode - tiek izpētītas, izprastas un aprakstītas metodes, kas var kalpot par daļu no problēmas risinājuma;
2. Salīdzinošā metode - aprakstītās metodes tiek salīdzinātas, tiek noteiktas to pozitīvās un negatīvās īpašības;
3. Analītiskā metode - tiek izpētīti un analizēti dažādi literatūras avoti.

# Problēmas raksturojums

## Problēmas sastāvs

Problēma sastāv no datorsistēmas lietotāja, galīgas un lokālas digitālas datu krātuves, kas var atrasties kādā personīgā datu nesējā vai arī uzņēmuma serverī. Datorsistēmas lietotājam ir nepieciešamība iegūt informāciju, kas atrodas kādā no datu krātuves dokumentiem. Par šo informāciju lietotājam kaut kas ir zināms, ne vienmēr tas ko lietotājs atcerēsies būs tāda paša tipa informācija. Tas varētu būt dokumenta metadati - izveidošanas datums, pēdējās rediģēšanas datums, dokumenta nosaukums, vai arī kāds virsraksts dokumentā, teksta fragments, kādas tabulas saturs vai tabulas lauku nosaukumi, attēls. Visbiežāk lietotājs aptuveni atcerēsies par ko ir tā teksta daļa kura satur nepieciešamo informāciju, tātad jāņem vērā ka meklējot lietotājs var ievadīt teksta fragmentu kurš tieši neatbildīs dokumentā esošajam.

Graphical user interface, diagram, Word

Description automatically generated

*1.1 Attēls* **Meklēšanas procesa entītiju klašu diagramma**

## Meklēšanas procesa apraksts

1. Lietotājs datorprogrammā ievada informāciju kuru viņš atceras par dokumentu, uzspiež pogu meklēt.
2. Veic atrasto rezultātu novērtēšanu, izlasa atgrieztus teksta fragmentus, iespējams atver un izlasa kādus dokumentus.
3. Ja lietotājs ir atradis nepieciešamo informāciju, tad beidz darbu.
4. Ja lietotājs nav atradis nepieciešamo informāciju, tad viņš atgriežas pie pirmā soļa un mēģina meklēt to pašu izmantojot citādāku vārdu kombināciju, vai meklēšanai pievienojot papildu informāciju.

Diagram

Description automatically generated

*1.2 Attēls* **Meklēšanas procesa aktivitāšu diagramma**



## Piemēri

Situācijas piemērs: It projekta vadītājs sapulcē ar izstrādes komandu apspriež klienta izmaiņu pieprasījumus. Esošajā situācijā dokumentācijas nav, bet ir nepieciešams ātri atrast ar klientu iepriekš izrunātas noteiktu prasību definīcijas lai saprastu vai tajās nebūs konflikti ar jaunajām. Iepriekšējās prasības pirms vairākiem mēnešiem kādā sapulcē ar klientu dokumentējis bija sistēmanalītiķis kurš šajā uzņēmumā vairs nestrādā.

Meklēšanas procesa piemērs (atbilst situācijas piemēram): projekta vadītājs meklēšanas programmā ieraksta tekstu “Kalendāra funkcijas definīcijas apraksts” novērtējot atgrieztos rezultātus saprot, ka atrasti daudzi kalendāra funkciju apraksti no citiem projektiem un sistēmu daļām (meklēts tiek uzņēmuma kopējā serverī), projekta vadītājs aptuveni zina, kad tika izveidots dokuments viņš pievieno šo datuma informāciju papildus teksta fragmentam. Meklē vēlreiz un viņam atgriezto vērtību saraksta augšgalā uzreiz izdodas ieraudzīt meklēto informāciju.

# Dokumentu indeksēšana



## Ievads

Informācijas meklēšanu atbilstoši ievadītam teksta fragmentam varētu veikt tā, ka meklējamajā datņu apgabalā katra datne tiek atvērta, tā tiek lasīta no sākuma līdz beigām un tiek noteikts vai tā satur teksta fragmentu. Šāda realizācija būtu pilnās pārlases pieeja, kas salīdzinot ar dokumentu indeksēšanas metodi meklēšanas procesam prasītu ievērojami vairāk laika. Ne tikai vajadzētu lasīt visas datnes no sākuma līdz beigām, bet arī tās vajadzētu atvērt, aizvērt un pārvietoties pa datņu struktūru.

Savukārt izmantojot dokumentu indeksēšanu būtu jālasa tikai speciāla datu struktūra, kurā ir izveidots kartējums katram meklējamajā datņu apgabalā esošajam vārdam ar datnes identifikatoru. Struktūrā esošie vārdi tiek sakārtoti alfabētiskā secībā un līdz ar to var pielietot bināro meklēšanu. Lai veicot meklēšanu varētu izmantot indeksētu struktūru ir nepieciešams to izveidot pirms meklēšana tiek veikta, kā arī lai lietotājam varētu atgriezt īsceļu uz datni, datnes identifikatoriem jābūt sakartētiem ar datnes adresi uz datu nesēja.

## Bag of words

### Struktūras apraksts

Dokumenti orģināli tiek veidoti cilvēku dabiskajā valodā, kas nozīmē, ka līdz ar vārdiem ir dažādas pieturzīmes, un dažādi teikumu struktūru veidojoši elementi kā atstarpes starp vārdiem, atkāpes, pārejas jaunā rindā u.tml. Tāpat dokumentiem ir dažādi citi satura izskata veidojoši elementi – teksta krāsa, izcelts teksts, atstarpju lielums, dažāda tipa grafiskie elementi, tabulas u.tml. Šīs lietas teksta fragmenta meklēšanai nav būtiskas, būtiski ir tikai vārdi un to secība.

Bag of words ir tāda dabiskā valodā pierakstīta satura reprezentācija, kurā netiek iekļauti visi iepriekš uzskaitītie liekie elementi. Šī struktūra dod iespēju daudzus dokumentus apvienot vienotā datu struktūrā kurā tiek glabāti tikai vārdi un to biežums, katrs vārds ir sakartēts ar attiecīgo dokumentu pēc identifikatora. Bag of words vienkāršā datu struktūra bez papildus metodēm pati par sevi jau uzlabo meklēšanas procesa izpildes ātrumu, jo daudzu dokumentu vietā jāapstaigā tikai viena struktūra un nav jāapstaigā daudzi meklēšanai lieki elementi. Toties ar šo struktūru vien nepietiek, lai varētu veikt saturīgu teksta fragmenta meklēšanu, jo tā neiekļauj informāciju par vārdu secību dokumentos. Šī vienkāršā teksta reprezentācija to padara daudz labāk mašīnlasāmu un arī tiek plaši izmantota dabiskās valodas apstrādē un dažādos mašīnmācīšanās algoritmos.

### Struktūras izveidošana

Var tikt veidota kā divdimensionāla struktūra kuras vienā dimensijā ir dokumentu identifikatori, un otrā dimensijā ir tajos esošie vārdi. Lai varētu izveidot Bag of words datu struktūru ir nepieciešams:

1. Apstaigāt visu datņu kolekciju kuras tekstu saturs tiks iekļauts datu struktūrā.
2. Katru datnē esošo vārdu ir nepieciešams atpazīt (nosakot vārda robežas) un izgūt.
3. Katru izgūto vārdu jāievieto struktūras vārdu dimensijā ja tas jau tur nav, ja ir tad jāpalielina tā biežumu dokumentā kurš dotajā brīdī tiek lasīts.

**Struktūras piemērs**

Piemērā tiek izmantoti 3 dokumenti, kur katram no tiem ir dots tā identifikators un saturs.

1 = “Mežs ir ekosistēma. Mežs ir arī barības avots.”

2 = “Mežs sastāv no daudz kokiem.”

3 = “Koki vispār ir mežu pamatā. Ēģiptē mežu nav vispār.”

### Realizācija divdimensionālā matricā

Piemēra attēlojums divdimensionālā matricā, kur katra rinda atbilst vienam vārdam un katra kolonna atbilst dokumenta vektoram, kurš nosaka kuri vārdi un cik lielā skaitā ir dokumentā.

* 1. *tabula*

**Bag of words realizācija divdimensionālā masīvā**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 2 | 3 |
| Mežs | 2 | 1 | 0 |
| ir | 2 | 0 | 1 |
| ekosistēma | 1 | 0 | 0 |
| arī | 1 | 0 | 0 |
| barības | 1 | 0 | 0 |
| avots | 1 | 0 | 0 |
| sastāv | 0 | 1 | 0 |
| no | 0 | 1 | 0 |
| daudz | 0 | 1 | 0 |
| kokiem | 0 | 1 | 0 |
| Koki | 0 | 0 | 1 |
| vispār | 0 | 0 | 2 |
| mežu | 0 | 0 | 2 |
| pamatā | 0 | 0 | 1 |
| Ēģiptē | 0 | 0 | 1 |
| nav | 0 | 0 | 1 |

**Secinājumi**

Šādā realizācijā sanāk glabāt diezgan daudz nulles, kas reprezentē vārda neesamību dokumentā, kas ir neefektīva realizācija jo par dokumentos neeksistējošiem vārdiem jāglabā informācija, kas ne tikai prasa papildus atmiņu, bet arī palēlina apstrādes ātrumu, jo ir jāapstaigā vairāk elementi. Vārdi nav sakārtoti un līdz ar to nevar implementēt efektīvu meklēšanu.

Vārdi tiek glabāti izmantojot UTF-16 kodējumu, kas nozīmē, ka viens simbols aizņem 2 baitus, un visi skaitļi tiek glabāti 4 baitos. Tātad atbilstoši dotajam piemēram (3 dokumenti un 16 unikālie vārdi) kopējais struktūras atmiņas apjoms ir 350 baiti.

Šīs struktūras atmiņas sarežģītība ir vērtējama kā O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir kopējais dokumentu skaits.

### Alternatīva realizācija divdimensionālā matricā

Šajā realizācijā tiek novērsta nuļļu glabāšanas problēma tādā veidā, ka vārda esamības fakts dokumentā tiek glabāts struktūrā kuras katrs elements ir divvietīgs, kura pirmajā laukā tiek glabāts dokumenta identifikators un otrajā laukā tiek glabāts atkārtošanās reižu skaits. Vārdi ir sakārtoti tā ka ir iespējams realizēt bināro meklēšanu. Visi vārdi tiek glabāti vienā formā – tikai ar mazajiem burtiem šādi arī potenciāli samazinot kopējo vārdu skaitu, lai piemēram nebūtu jāglabā atsevišķi vārdi “mežs” un “Mežs”. Ja reģistrjutība tiek ievērota, tad katrs teikuma pirmais vārds ir papildus vārds struktūrā, kas aizņem papildus resursus.

Sekojošajā alternatīvās realizācijas attēlā izmanto to pašu dokumentu kolekciju, kas tiek izmantota iepriekšējā struktūras piemērā.

Table

Description automatically generated

*2.1 Attēls* **Bag of words alternatīva realizācija**

**Secinājumi**

Reālā datu nesējā būs daudz vairāk dokumenti kā piemērā līdz ar to arī daudz vairāk nulles būtu bijušas un papildus dokumenta identifikatora glabāšana atmaksāsies gan atmiņas ziņā, gan arī apstrādes ziņā (mazāk elementi apstrādei). Bināro meklēšanu vārdu dimensijai var implementēt tā, ka atrod vārdu tikai pēc pirmā burta un tad turpina ar pilno pārlasi, vai bināro meklēšanu veic secīgi katram vārda burtam. Binārā meklēšana ļoti ievērojami uzlabo jebkura vārda atrašanas ātrumu, kas ir O(log n), bet šeit jāņem vērā, ka binārajai meklēšanai būs papildus operācijas jāveic jo darbs ir ar simbolu virknēm nevis skaitļiem (salīdzinot ar absolūtās vērtības realizāciju).

Visi vārdi tiek glabāti izmantojot UTF-16 kodējumu (2 baiti simbolam), un visi skaitļi tiek glabāti 4 baitos. Tātad atbilstoši tam pašam dotajam piemēram (3 dokumenti un 16 unikālie vārdi) kopējais struktūras atmiņas apjoms ir 302 baiti.

Šīs struktūras atmiņas sarežģītība ir vērtējama kā O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir dokumentu skaits kuros attiecīgais vārds eksistē.

### Absolūtās vērtības realizācija

Simbolu virknes varētu glabāt nevis to orģinālajā formātā, kas sastāv no secīgiem burtiem, bet izmantojot noteiktu algoritmu varētu katram vārdam iegūt tikai tam atbilstošu unikālu skaitli. Citādi šī realizācija ne ar ko neatšķiras no iepriekšējās, vērtības ir sakārtotas un iegūtas tikai no mazajiem burtiem.

Svarīgi ir lai algoritms nodrošina, ka katram vārdam atbildīs tieši viens unikāls skaitlis arī tad, ja vārds ir tāda simbolu virkne kuru lasot no abām pusēm sanāk atsevišķi valodai piederīgi vārdi, piemēram vārdiem alus un sula katram jābūt savam unikālajam skaitlim.

**Vērtības iegūšanas algoritma pseidokods**

Summai piešķir 0

Reizinātājam piešķir 1

Kamēr nav apstaigāti visi vārda simboli

Vārda simbola decimālo vērtību reizina ar reizinātāja vērtību un tad pieskaita summai

Reizinātāja vērtību palielina par 0.5

Cikla beigas

Atgriež summu

**Secinājumi**

Vērtības kas iegūtas izmantojot šo algoritmu ar lielu rezervi saies 4 baitos. Gan vārda alus, gan vārda sula simbolu kopējā decimālā summa ir 437 (Izmantojot UTF-16 kodējumu). No algoritma iegūtā vērtība vārdam alus ir 780 un sula ir 748. Algoritma laika sarežģītība ir O(n), kur n ir vārda simbolu skaits.

**Piemērs**

866 = arī, 1131 = avots, 1955 = vispār, 1964 = sastāv.

Diagram

Description automatically generated

*2.2 Attēls* **Absolūtās vērtības realizācijas piemērs**

**Secinājumi**

Ja simbolu virkne UTF-16 kodējumā vidēji varētu aizņemt ap 20 baitiem (viens simbols divi baiti), tad šādu skaitlisku vērtību ar lielu rezervi varētu glabāt četros baitos (int tipā) un līdz ar to arī kopējais struktūras izmērs varētu ievērojami samazināties. Pie šādas pieejas arī binārā meklēšana strādās efektīvāk, jo ar skaitļiem operēt būs daudz ātrāk nekā ar simbolu virknēm. Tomēr ja struktūrā tiek glabāta tikai absolūtā vērtība, tad tas var radīt ierobežojumus tālākai apstrādei.

Katram vārdam tiek glabāts tam atbilstošs unikāls skaitlis (4 baiti), un tāpat dokumentu identifikatori un tajos esošo vārdu skaits tiek glabāts 4 baitos. Tātad atbilstoši tam pašam dotajam piemēram (3 dokumenti un 16 unikālie vārdi) kopējais šīs struktūras atmiņas apjoms ir 208 baiti.

Šīs struktūras atmiņas sarežģītība ir vērtējama kā O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir dokumentu skaits kuros attiecīgais vārds eksistē.

### Bag of words nobeiguma secinājumi

Meklēšanas sistēmas realizēšanai ir vērts izmantot vairākus Bag of words dokumentu reprezentācijas principus: lieko elementu (pieturzīmes, izskatu veidojoši elementi) atmešana; visos dokumentos esošo vārdu apkopošana vienotā struktūrā (kartējumā) kurā nebūs to dublikāti, bet būs informācija par biežumu; neglabāt vārda neesamības faktu struktūrā; vārdu sakārtošana, lai varētu realizēt efektīvu meklēšanu. Bet lai implementētu efektīvu meklēšanas sistēmu ar Bag of words sniegtajām iespējām ir stipri par maz, piemēram nav iespējas meklēt frāzes, jo netiek netiek glabāta informācija par vārda atrašanās vietu dokumentā (nevar noteikt kuri vārdi kādā attālumā atrodas viens no otra dokumentā).

## Inverted Index

### Pamatvarianta struktūras apraksts

Tipiskā meklēšanas scenārijā lietotājs meklēs frāzi nevis vienu individuālu vārdu, tāpēc Bag of words pieeja nesniedz pilnīgas iespējas meklēšanas sistēmas realizēšanai. Savukārt Inverted Index ir datu struktūra, kas ietver sevī Bag of words teksta reprezentācijas pozitīvos principus, un papildina tos galvenokārt ar informāciju par vārda atrašanās pozīciju dokumentā, tādējādi katrs kolekcijas vārds ir sakartēts ar visiem dokumentiem kuros tas atrodas un ar visām tā atrašanās pozīcijām. Tādējādi tiek dota iespēja meklēt arī konkrētas frāzes apstaigājot indeksa struktūru. Pozīciju glabāšana ļauj ne tikai atrast secīgas frāžu sastāvdaļas, bet arī dod iespēju atrast vārdus starp kuriem ir noteikts attālums, piemēram 3 vārdi starpā starp vienu un otru meklēto.

Ņemot vērā to ka lietotājs meklējot noteiktu informāciju, ne vienmēr to atcerēsies vai galvā iedomāsies tieši tā, kā tas dokumentā vienā versijā ir aprakstīts, ir svarīgi iestrādāt mehānismu, kas ļauj atrast nepieciešamo informāciju arī ja lietotājs nav precīzu teksta fragmentu uzrakstījis. Risinājums ir vārdus glabāt nevis tieši tādā formā kādā tie dokumentā atrodas, bet glabāt tos kā lemmas. Latviešu valodā tas darbības vārdiem būtu nenoteiksme un lietvārdiem nominatīvs. Lietotāja ievadīto teksta fragmentu arī pārveido lemmas formā un tādējādi varēs atrast informāciju arī tad ja lietotājs nebūs ievadījis precīzu teksta fragmentu.

### Struktūras izveidošana

Tāpat kā Bag of words pieejā šī datu struktūra ir jāizveido pirms meklēšana tiek veikta. Struktūra tiek veidota kā divdimensionāla, jāņem vērā ka uzsākot tās izveidošanu nav zināms cik vārdi un to pozīcijas dokumentos būs jāievieto tāpēc tās realizācijā jāizmanto tādas datu struktūras kurām nav iepriekš jāzina garums. Vienā dimensijā ir visi dokumentos esošie vārdi (bez dublikātiem) un otrā dimensijā šī vārda pozīcija katrā dokumentā kurā tas atrodas. Lai struktūru izveidotu jāveic sekojoši soļi:

1. Apstaigāt visu datņu kolekciju kuras tekstu saturs tiks iekļauts datu struktūrā.
2. Katru datnē esošo vārdu ir nepieciešams atpazīt (nosakot vārda robežas) un secīgi izgūt.
3. Izgūtais vārds ir jāpārveido uz mazajiem burtiem, jāaizvāc no tā jebkas kas nav burts un tad jāpārveido uz lemmas formu.
4. Ja vārds lemmas formā jau ir vārdu sarakstā, tad dokumenta identifikators un vārda pozīcija dokumentā kurš tiek apstrādāts jāievieto attiecīgā vārda pozīciju sarakstā tā, lai pēc pievienošanas tas būtu sakārtots pēc dokumenta identifikatoriem augošā secībā.
5. Ja vārds lemmas formā jau nav vārdu sarakstā, tad tas lemmas formā ir jāpievieno vārdu sarakstā attiecīgajā vietā tā lai pēc vārda pievienošanas vārdu saraksta dimensijas saturs būtu saranžēts.

**Struktūras piemērs**

Piemērā tiek izmantoti 3 dokumenti, kur katram no tiem ir dots identifikators un tā saturs.

1 = “Mežs ir ekosistēma. Mežs ir arī barības avots.”

2 = “Mežs sastāv no daudz kokiem. Meža ekosistēmā dzīvo dažādi organismi.”

3 = “Koki vispār ir mežu pamatā. Ēģiptē mežu nav vispār.”

Histogram

Description automatically generated

*2.3 Attēls* **Inverted Index datu struktūras realizācijas piemērs**

**Secinājumi**

Vārdu transformēšana lemmu formā ne tikai ļauj lietotājam atrast frāzes kuras ietver vārdus citādās gramatiskajās formās, bet arī samazina indeksa struktūrā kopējo esošo vārdu skaitu, tādējādi taupot atmiņu un uzlabojot apstrādes laiku. Jāņem vērā ka attēlotais piemērs ir relatīvi mazs, un reālā dokumentu indeksa struktūrā vārdi atkārtosies daudz biežāk. Svarīgi arī pozīciju saraksta struktūru uzturēt sakārtotu pēc dokumenta indeksiem un pozīcijas, lai varētu efektīvāk apstrādāt arī šo daļu.

Vārdi tiek glabāti izmantojot UTF-16 kodējumu, kas nozīmē, ka viens simbols aizņem 2 baitus, un visi skaitļi tiek glabāti 4 baitos. Tātad atbilstoši dotajam piemēram (3 dokumenti un 21 unikālais vārds (orģinālformā)) kopējais struktūras atmiņas apjoms ir 390 baiti.

Šīs struktūras atmiņas sarežģītība ir vērtējama kā O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits lemmas formā un m ir kopējais attiecīgo vārdu pozīciju skaits visos dokumentos.

### Inverted Index ar metadatiem

Lai meklēšanas iespējas padarītu plašākas ir jādod neobligāta iespēja lietotājam pievienot papildus informāciju teksta fragmentam. Situācijās kur lietotājs slikti spēj iedomāties vai atcerēties teksta fragmentu, viņš varētu uzrakstīt īsāku frāzi, bet meklēšanai pievienot papildus meta informāciju par to kurš dokumentu ir veidojis, vai kad dokuments ir pēdējo reizi rediģēts u.tml. Ja meklēšana tiek veikta gan pēc teksta fragmenta, gan pēc metadatiem, tad rezultāts tiek veidots kā šķēlums no šīm abām rezultātu kopām, līdz ar to ja meklēšanas rezultātu kopa ir plaša, tad izmantojot papildus meta informāciju lietotājs varētu šo pašu kopu sašaurināt un nogriezt liekos variantus nost. Varbūt lietotājs vēlas atrast dokumentus tikai pēc meta informācijas piemēram visi dokumenti, kas veidoti noteiktā laika periodā un kurus veidojusi noteikta persona vai personu kopa arī šādai iespējai vajadzētu būt.

### Struktūras izveidošana

Tiek veikti tie paši soļi, kas parastā inverted index veidošanā, tikai par katru dokumentu kurš tiek iekļauts struktūrā tiek ievākti tā metadati un šie metadati ir jāiekļauj inverted index struktūrā. Dokumenta metadati sastāv no dažādiem atribūtiem un to vērtībām, kā arī vērtību tipiem. Takā ir zināms, ka sarakstā esošajiem vārdiem nebūs klāt citas interpunkcijas zīmes, jo tās tiek aizvāktas vārda izgūšanas procesa laikā, tad lai varētu atšķirt metadatu identifikatorus no dokumentu satura vārdiem metadatu identifikatoriem priekšā tiek pielikta izsaukuma zīme un tas sākas ar lielo burtu. Atlikusī metadatu identifikatora daļa sastāv no atribūta nosaukuma.

Tātad tā pati inverted index struktūras vārdu dimensija tiek papildināta ar metadatu identifikatoriem, bet tie visi atradīsies blakus viens otram. Dimensija kas izriet no katra metadatu identifikatora saturēs sarakstu ar trīsvietīgiem elementiem, kur pirmais lauks saturēs dokumenta identifikatoru, otrais lauks saturēs datu tipa nosaukumu, un trešais lauks datu tipa vērtību. Metadatu identifikatori sarakstā ir jāiekļauj pirms tiek no dokumentiem iegūtas to vērtības.

Līdz ar to inverted index struktūras veidošana tiek papildināta ar vienu soli, kas notiek pirms vārdu ielasīšanas: tiek ielasītas dokumenta metadatu vērtības un tās iepriekš minēto trīsvietīgo elementu formā tiek pievienotas metadatu identifikatoru sarakstos tā, lai tos uzturētu sakārtotus pēc atribūta vērtības augošā secībā. Jāielasa sekojošas metadatu vērtības: Title, Author, Company, CreateTime, LastSavedBy, LastSavedTime, NameOfApplication (definē datnes tipu).

**Struktūras piemērs**

Piemērā tiek izmantoti 3 dokumenti, kur katram no tiem ir dots identifikators un tā saturs, un tā metadati. Realitātē būs vairāk metadatu kā piemērā, piemēra mērķis ir reprezentēt principu.   
Šis piemērs tiek veidots uz iepriekšējā bāzes pievienojot tikai papildus metadatu informāciju.

1 = “Mežs ir ekosistēma. Mežs ir arī barības avots.” Metadati: Author{ Type: “String”, Value: "Jānis Bērziņš"}, Company{ Type: “String”, Value: "Best solutions" }, CreateTime{ Type: “String”, Value: 10/25/2021 }

2 = “Mežs sastāv no daudz kokiem. Meža ekosistēmā dzīvo dažādi organismi.” Metadati: Author { Type: “String”, Value: "Līga Kļaviņa" }, Company{ Type: “String”, Value: "Real solutions" }, CreateTime{ Type: “DateTime”, Value: 06/21/2021}

3 = “Koki vispār ir mežu pamatā. Ēģiptē mežu nav vispār.” Metadati: Author{ Type: “String”, Value: "Jānis Bērziņš" }, CreateTime { Type: “DateTime”, Value: 05/15/2021 }

Graphical user interface, application, table

Description automatically generated

*2.4 Attēls* **Inverted Index ar metadatiem datu struktūras realizācijas piemērs**

**Secinājumi**

Metadatu ievietošana indeksa struktūrā palielinās tā apjomu, bet neietekmēs vārdu atrašanas ātrumu, jo visi metadatu identifikatori atrodas blakus un to skaits ir konstants un attiecībā pret vārdu skaitu būs ļoti mazs. Meklēšanā izmantojot gan teksta fragmentus, gan metadatus lietotājam tiek pavērtas plašākas iespējas kā atrast vēlamo informāciju.

Visi vārdu un metadatu identifikatoru un to vērtību sarakstu simboli tiek glabāti izmantojot UTF-16 kodējumu, kas nozīmē, ka viens simbols aizņem 2 baitus, un visi skaitļi gan pozīciju sarakstos, gan metadatu vērtību sarakstos tiek glabāti 4 baitos. Tātad atbilstoši dotajam piemēram (3 dokumenti, 21 unikālais vārds (orģinālformā), 3 metadatu identifikatori ar to vērtību sarakstiem) kopējais struktūras atmiņas apjoms ir 764 baiti.

Šīs struktūras atmiņas sarežģītība ir vērtējama kā O(x\*y+n\*m), kur x ir struktūrā iekļauto metadatu kategoriju skaits, un y ir metadatu ierakstu skaits, bet savukārt n ir dokumentu kolekcijā esošo unikālo vārdu skaits lemmas formā un m ir kopējais attiecīgo vārdu pozīciju skaits visos dokumentos.

## Struktūras uzturēšana

Sākotnēji izveidojot meklēšanai nepieciešamo datu struktūru tā atbildīs reālajai situācijai, tam saturam, kas dokumentu kolekcijā ir uz to brīdi. Savukārt līdzko dokumentu kolekcijā tiks veiktas kādas izmaiņas, ir liela iespēja, ka izveidotā struktūra vairs nebūs atbilstoša reālajai situācijai, un tas potenciāli var novest pie neprecīziem meklēšanas rezultātiem. Tātad neatkarīgi no izmantotās metodes laika gaitā būs nepieciešamība to uzturēt.

Lai veicot struktūras atjaunošanu nebūtu jāpārbauda vai visi dokumenti vēljoprojām atbilst tai, ir jābūt izveidotai atsevišķai datu struktūrai kurā tiek glabāta informācija par to, kādā datumā dokuments datu struktūrā (meklēšanas) pēdējo reizi ir atjaunots, šai informācijai ir jābūt sakartētai ar dokumentu identifikatoriem.

Programmas saskarnē lietotājam jāsniedz informācija par to vai dokumentu kolekcijā ir tādi dokumenti kuru pēdējās rediģēšanas datumi ir jaunāki par datu struktūrā esošajiem, un jāsniedz iespēja atjaunot struktūru. Meklēšanā izmantoto datu struktūru atjaunojot nepieciešams ir atjaunot informāciju tikai par tiem dokumentiem kuru pēdējās rediģēšanas datumi ir jaunāki par datumu kartējuma struktūrā esošajiem.

## Kopsavilkums par dokumentu indeksēšanu

* 1. *tabula*

**Indeksēšanas datu struktūru kopsavilkums**

|  |  |  |  |
| --- | --- | --- | --- |
| Metode | Pozitīvie aspekti | Negatīvie aspekti | Atmiņas sarežģītības novērtējums |
| 1. Bag of words (Kopīgais) | * Vieglāk realizēt * Vieglāk atjaunot datu struktūru (Ja bijušas izmaiņas dokumentu kolekcijā) * Vienkāršām meklēšanas vajadzībām varētu būt pietiekami | * Nedod iespēju meklēt frāzi * Nav iespēja atrast vārdus citos locījumos (attiecībā pret lietotāja ievadītajiem vārdiem) * Nav izvērstas meklēšanas iespējas | - |
| 1.1 Realizācija divdimensionālā matricā (pamatvariants) | * Mazai dokumentu kolekcijai meklēšanas ātrums būtu pietiekošs * Konstantā laikā O(1) var noteikt ka vārds neeksistē dokumentā | * Datu struktūrā tiek aizņemta atmiņa arī ja vārds neeksistē dokumentā * Nav iespējas implementēt efektīvu meklēšanu | O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir kopējais dokumentu skaits |
| 1.2 Alternatīva realizācija divdimensionālā matricā | * Aizņem mazāk atmiņu salīdzinot ar pamatvariantu * Var implementēt bināro meklēšanu, līdz ar to arī lielai dokumentu kolekcijai būs labs meklēšanas ātrums | * Sarežģītāk noteikt vai konkrēts vārds neeksistē dokumentā | O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir dokumentu skaits kuros attiecīgais vārds eksistē |
| 1.3 Absolūtās vērtības realizācija | * Aizņem vismazāk atmiņu no visām apskatītajām metodēm (Glabā tikai skaitļus) * Labāks meklēšanas ātrums (Jāmeklē skaitļos ne simbolu virknēs) | * Neglabā informāciju par vārda orģinālo sastāvu * Vairāk laiks vajadzīgs lai noteiktu vai vārds neeksistē dokumentā | O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits un m ir dokumentu skaits kuros attiecīgais vārds eksistē |
| 2. Inverted index (kopīgais) | * Sniedz iespēju meklēt konkrētu frāzi. * Var meklēt vārdus ar noteiktu attālumu starp tiem * Struktūrā atradīs frāzi arī ja tās vārdiem atšķirsies locījumu forma no lietotāja ievadītās | * Aizņem vairāk atmiņu par Bag of words metodēm. * Sarežģītāka realizācija un uzturēšana | - |
| 2.1 Pamatvariants | * Aizņem mazāk atmiņu (Salīdzinot ar 2.2) * Vieglāk realizēt (Salīdzinot ar 2.2) | * Nav iespējas meklēšanā iesaistīt metadatus | O(n\*m), kur n ir dokumentu kolekcijā esošo unikālo vārdu skaits lemmas formā un m ir kopējais attiecīgo vārdu pozīciju skaits visos dokumentos |
| 2.2 Metadatu variants | * Plašākas meklēšanas iespējas tajā iekļaujot metadatus | * Aizņem vairāk atmiņu (Salīdzinot ar 2.1) * Sarežģītāka realizācija | O(x\*y+n\*m), kur x ir struktūrā iekļauto metadatu kategoriju skaits, un y ir metadatu ierakstu skaits, bet savukārt n ir dokumentu kolekcijā esošo unikālo vārdu skaits lemmas formā un m ir kopējais attiecīgo vārdu pozīciju skaits visos dokumentos. |

# Dokumentu meklēšana un atgriešana

Indeksēšanas sadaļā skaidrs kļūst tas, ka dokumentu meklēšanas process notiks iepriekš sagatavotā datu struktūrā – inverted index ar metadatiem, šī datu struktūra dod iespējas veikt efektīvu meklēšanu, un jāsaprot ir kā šīs iespējas pareizi izmantot, lai izmantotu tās pilnu potenciālu. Tātad meklēšanas procesa rezultāts ir atkarīgs no izveidotās datu struktūras un veida kā tā tiks izmantota, bet ne tikai. Būtiski arī ir interpretēt lietotāja sniegtos ievaddatus tā lai varētu tam atgriezt vēlamo rezultātu.



## Kopīgais piemērs

Lai varētu pēc iespējas objektīvāk salīdzināt un izvērtēt tālāk apskatītās metodes tiek ieviesta viena dokumentu kopa ar noteiktu saturu, kura tiks lietota tālāk esošajās metodēs (kurās tas ir iespējams un nepieciešams). Tiek pieņemts, ka vārdi datu struktūrā kurā veic meklēšanu ir lemmas formā, tāpat arī lietotāja ievadītie vārdi tiek konvertēti uz lemmas formu tādējādi nodrošinot, ka tiks atrasti vārdi arī citos locījumos. Kopumā ir 5 dokumenti.

*3.1 tabula*

**Kopīgā piemēra dokumenti un to saturs**

|  |  |
| --- | --- |
| Identifikators | Saturs |
| 1 | Mežs ir bagātīga ekosistēma plašā teritorijā. Mežs ir arī barības avots. |
| 2 | Mežs sastāv no daudz kokiem. Meža ekosistēmā dzīvo dažādi organismi. |
| 3 | Koki vispār ir mežu pamatā. Ēģiptē mežu nav vispār. |
| 4 | Daļa meža ekosistēmas atrodas augsnē esošajās baktērijās. |
| 5 | Par priežu mežiem ir zināms, ka tie ir mūžzaļi. |

## Meklēšanas algoritmi

### Levenšteina algoritms

Tiek izmantots, lai noteiktu attālumu starp divām simbolu virknēm, kur attālums raksturo to cik līdzīgas ir simbolu virknes. Jo lielāks ir attālums starp virknēm, jo atšķirīgākas tās ir pēc to simboliem. Attālums starp simbolu virknēm tiek mērīts skaitot to cik operācijas ir nepieciešams veikt, lai no vienas simbolu virknes iegūtu otru, tātad jo vairāk operācijas jo lielāks attālums. Operācijas var būt aizvietot, pievienot vai dzēst simbolu tās virknē.

Algoritms ir veidots tā, ka katrā solī tas atrisina problēmas apakšproblēmu balstoties uz iepriekšējās atrisinātās apakšproblēmas rezultātiem, katrā solī izvēloties mazāko iespējamo operāciju skaitu lai no vienas simbolu virknes varētu iegūt otru. Tā tiek risinātas apakšproblēmas kamēr tiek iegūts globāls risinājums, īsākais ceļš kā no vienas simbolu virknes var iegūt otru.

Var būt dažādi lietojumi, piemēram kļūdu labošanai, lietotāja ievadītajam vaicājumam tiek atrasta ļoti līdzīga simbolu virkne, no līdzīgajām atlasot viss līdzīgāko (ar mazāko attālumu) un tad šo simbolu virkni var piedāvāt lietotājam, varbūt viņš vēlas pamēģināt meklēt ar to. Var pielietot arī, lai iekļautu meklēšanas rezultātos dokumentus, kas iekļauj ļoti līdzīgas virknes, šis var būt noderīgi situācijā kur lietotājs ir pieļāvis kādu gramatikas kļūdu, vai arī nedaudz citādāk noformulējis savu informācijas vajadzību.

**Piemērs**

Algoritma izpilde tiek nodemonstrēta nosakot attālumu divām simbolu virknēm: “Mežs ir avots” un “Mezs arī avots”. Pieņemot ka otrā virkne varētu būt ievadīta no lietotāja ar gramatikas kļūdu un vārda vietā “arī” tiek lietots vārds “ir”, kas ir diezgan reāla situācija, jo lietotājs savu informācijas vajadzību var noformulēt nedaudz citādāk nekā tā tiešā tekstā atrodas dokumentā.

Table

Description automatically generated

*3.1 Attēls* **Levenšteina algoritma izpildes attēlojums**

**Secinājumi**

Algoritma laika sarežģītība ir O(x\*y), kur x un y ir simbolu virkņu garumi. Tāpat arī papildus atmiņa, kas ir jāizmanto lai varētu realizēt algoritmu ir O(x\*y), kur x un y ir simbolu virkņu garumi. Simbolu virkņu līdzības noteikšana var noderēt meklēšanas sistēmā, bet jāņem vērā, ka tās aprēķināšana prasa zināmus resursus un jāseko līdzi kā šī algoritma iekļaušana kopējā meklēšanas realizācijā ietekmē tās kopējo izpildes laiku.

### Binārā meklēšana un inverted index

Binārā meklēšana Ievērojami uzlabo meklēšanas ātrumu, ja pilnās pārlases meklēšanā katrā solī apgabals kurā elements vēl tiks meklēts samazinās par vienu vienību, tad izmantojot binārās meklēšanas principu, katrā solī šis apgabals samazinās uz pusi. Tāpēc ļoti vēlams ir izveidot indeksa struktūru tā lai meklēšanu tajā var realizēt izmantojot bināro meklēšanu. Lai bināro meklēšanu efektīvi varētu izmantot visiem elementiem struktūrā ir jābūt sakārtotiem un elementiem jāvar uzreiz piekļūt pēc indeksa.



### Binārās meklēšanas algoritms

Tiek lietots klasiskais binārās meklēšanas algoritms inverted index datu struktūras vārdu dimensijas vārdu meklēšanai. Vārdiem jābūt sakārtotiem alfabētiskā secībā, katrā izpildes solī jāvar noteikt vai meklētais vārds ir lielāks vai mazāks par vidējo vārdu. Piemēri un secinājumi ir nākamajā sadaļā, jo binārā meklēšana tiek iekļauta kopējā meklēšanas procesā.

Kamēr augšējais indekss lielāks par apakšējo indeksu

Vidējam elementam piešķir augšējā un apakšējā elementu summas dalījumu ar divi.

Ja meklētais vārds ir vienāds ar vidējā elementa vārdu

Atgriež atrasto vārdu (beidz darbu)

Ja meklētais vārds lielāks par vidējā elementa vārdu

Apakšējam indeksam piešķir vidējā elementa indeksu

Ja meklētais vārds mazāks par vidējā elementa vārdu

Augšējam indeksam piešķir vidējā elementa indeksu

Cikla beigas

### Inverted index meklēšanas algoritms

Aprakstīts tiek meklēšanas pamat algoritms priekš inverted index datu struktūras. Skaitļošana sākas ar jau iegūtas lietotāja ievadītās simbolu virknes apstrādi. Pointeru saraksts glabā visu atrasto vārdu pozīciju sarakstu sākuma un beigu indeksus, kuri atbilst kopējam pozīciju sarakstam, katram vārdam ir sava apakš pozīciju saraksta daļa kopējā pozīciju sarakstā. Kopējais pozīciju saraksts apstrādāts tiek sinhroni ar pointeru sarakstu. Tālākais apraksts tiek dots augsta līmeņa pseidokoda formā, lai varētu iegūt priekštatu par būtiskākajiem algoritma darbības soļiem, nevis katru realizācijas detaļu.

Kamēr tiek iegūts nākamais vārds no ieejas simbolu virknes

Vārdu pārveido uz mazajiem burtiem

Vārdu pārveido lemmas formā

Ja izmantojot bināro meklēšanu vārds tiek atrasts indeksa struktūras vārdu dimensijā

No indeksa struktūras iegūst atrastā vārda pozīciju sarakstu

Iegūto pozīciju sarakstu pievieno kopējā pozīciju saraksta beigās

Pointeru sarakstam pievieno jaunu elementu- pozīciju saraksta sākuma un beigu indeksus

Citādi ja netika atrasts

Atgriež paziņojumu, ka neviens dokuments kolekcijā nesatur šādu frāzi

Cikla beigas

Kamēr kāda kopējā pozīciju saraksta daļa nav apstaigāta līdz beigām

Ja kopējā pozīciju sarakstā ir atrasta tāda elementu kopa, ka visās pozīciju saraksta daļās sākot no pirmās ir elements kuram dokumenta identifikators ir kopīgs ar visiem pārējiem kopas elementiem, un katrā nākamajā daļā elementa vārda pozīcija ir par vienu vienību lielāka

Dokumenta identifikators tiek pievienots atrasto dokumentu sarakstā.

Cikla beigas

Tiek atgriezts atrasto dokumentu saraksts

**Piemērs**

Tiek meklēts teksta fragments “Mežs ir”, inverted index struktūrā visi vārdi glabājas lemmas formā ir sakārtoti augošā secībā, tāpat arī pozīciju saraksti ir sakārtoti augošā secībā pēc dokumentu identifikatoriem. No kopējā piemēra struktūrā tiek iekļauti dokumenti 1 un 4. Teksta fragmenta vārdi pirms meklēšanas tiek pārveidoti lemmas formā.

Sākumā tiek atrasts vārds “mežs”:

Diagram

Description automatically generated with low confidence

*3.2 Attēls* **Izpildīta binārā meklēšana vārdam mežs inverted index datu struktūrā**

Tad tiek atrasts vārds “būt”:

Diagram, table

Description automatically generated

*3.3 Attēls* **Izpildīta binārā meklēšana vārdam mežs inverted index datu struktūrā**

Tālāk tiek veikta frāzes atrašana pozīciju sarakstos:

Diagram

Description automatically generated

*3.4 Attēls* **Izpildīta frāzes atrašana kopējā pozīciju sarakstā**

**Secinājumi**

Binārā meklēšana ļauj ļoti efektīvi atrast visus vārdus iekš vārdu dimensijas laikā O(log x), kur x ir elementu skaits masīvā. Kopējā pozīciju saraksta apstaigāšana notiek laikā O(y), kur y ir visu pozīciju (visu meklējamo vārdu) sarakstu kopējais garums. Lietotāja ievadītās frāzes apstrāde notiek laikā O(z), kur z ir lietotāja ievadītās simbolu virknes garums. Tātad kopējā laika sarežģītība ir O(z \* log x + y).

Kopējais pozīciju saraksts prasa papildus O(y) atmiņu, kur y ir visu vārdu pozīciju sarakstu kopējais garums, un pointeru saraksts prasa papildus O(w) atmiņu, kur w ir lietotāja ievadīto vārdu skaits. Lai arī izpildes laikā kopējais pozīciju saraksts prasīs O(y) operatīvo atmiņu, kopumā tas nebūs ļoti liels atmiņas patēriņš, jo katrs pozīciju saraksta elements sastāv tikai no diviem skaitļiem, kurus atkarībā no situācijas un implementācijas var glabāt 2 vai 4 baitos.

Ja vārdi ir sakārtoti pēc absolūtās vērtības un tā tiek iekļauta struktūrā, tad bināro meklēšanu vārdiem var veikt pēc absolūtās vērtības, kas prasīs mazāku skaitļošanas laiku. Kopumā šāda laika ziņā efektīva meklēšanas realizācija dod iespēju iestrādāt dažādas papildus iespējas (tērēt laiku citām operācijām), lai meklēšanu padarītu efektīvāku rezultāta ziņā.

## Vērtību iekļaušanas modelis

**Apraksts**

Meklēšana pamatā tiek balstīta uz lietotāja ievadītai frāzei atbilstošas meklēšanu dokumentu kopā, un tie dokumenti, kas satur lietotāja ievadīto frāzi tiks atgriezti. Pieļauj arī metadatu lietošanu meklēšanā un iespēju veikt meklēšanu kombinējot metadatus un tekstuālas frāzes. Ja dati ir strukturēti pa entītijām, tad meklēšana var tikt veikta attiecīgi noteiktai vai noteiktām entītijām.

Priekšrocības:

* Salīdzinājumā ar boolean meklēšanas modeli lietotājam nav nepieciešamas papildus zināšanas veicot meklēšanu;
* Vienkāršāka un ātrāka realizācija;
* Daudzos gadījumos, lai lietotājs realizētu savu informācijas vajadzību pietiek ar šo modeli un sarežģītāki nav nepieciešami.

Negatīvie aspekti:

* Lietotājam ir ierobežotas iespējas, nav tādas papildiespējas paplašināt savu vaicājumu, kā boolean tipa modelī;
* Nepieļauj rezultātu sakārtošanu pēc atbilstības lietotāja vajadzībai, nosaka tikai to vai iekļauj lietotāja sniegtās vērtības vai nē;
* Bieži vien tiks atgriezti stipri par daudz rezultāti un lietotāja informācijas vajadzība būs kaut kur pa vidu lielā rezultātu kopā, kas ir nopietni negatīvs faktors.

**Struktūra**

Tiek demonstrēts veids, kā lietotājs bez jebkādām papildus zināšanām var strukturēt savu meklēšanas vaicājumu. Šajā variantā lietotājs var meklēt dokumentus pēc frāzes un papildus norādīt informāciju par dokumenta metadatiem. Salīdzinot ar citiem meklēšanas modeļiem nesniedz nekādas izvērstas iespējas, bet tomēr var veikt precīzāku meklēšanu nekā tikai pēc frāzes.

Graphical user interface, text, application, email

Description automatically generated

*3.5 Attēls* **lietotāja saskarne nodrošina meklēšanu pēc metadatiem un frāzes**

**Piemēri**

Kopīgais piemērs tiek papildināts ar metadatiem:

Id = 1, Author{ Type: “String”, Value: "Jānis Bērziņš"}"}, Company{ Type: “String”, Value: "Best solutions" };

Id = 3, Author{ Type: “String”, Value: "Edgars Kļaviņš"}, CreateTime{ Type: “DateTime”, Value: 10/10/2020};

Id = 4, Author{ Type: “String”, Value: "Artūrs Lapiņš"}, Company{ Type: “String”, Value: "Best solutions" }.

Lietotāja ievadīta izteiksme: “Mežs ir”

Lietotāja ievadīti metadati: Author = “Jānis Bērziņš”

No kopīgā piemēra atgrieza: 1

Lietotāja ievadīta izteiksme: “Meža ekosistēmā dzīvo”

Lietotāja ievadīti metadati: Author = “Jānis Bērziņš”

No kopīgā piemēra atgrieza: -

Lietotāja ievadīta izteiksme: “Ekosistēma atrodas”

Lietotāja ievadīti metadati: Company = “Best solutions”

No kopīgā piemēra atgrieza: 4

## Patiesumvērtību meklēšanas modelis

**Apraksts**

Šajā modelī lietotāja ievaddati satur patiesumvērtības (boolean) tipa izteiksmi, kas tiek veidota no vārdiem un loģiskajiem operatoriem. Tiek lietoti loģiskie operatori kā: and, or, not, kā arī dažādas citas papildiespējas atkarībā no varianta. Lietotāja sniegtā izteiksme tiek iegūta simbolu virknes formā, līdz ar to pirms tās izpildes ir nepieciešamība to interpretēt. Lai no lietotāja saņemto izteiksmi varētu veiksmīgi interpretēt tai ir jābūt standartizētā formā, tādā ko programma spēs atpazīt.

Galvenās priekšrocības:

* Šo modeli ir relatīvi viegli implementēt sistēmā;
* Izmantojot šo modeli ir iespējams ļoti precīzi atrast tieši to, kas lietotājam ir vajadzīgs;
* Sniedz lietotājam nosacītu kontroli pār rezultātu kopu (Viegli paplašināt vai samazināt rezultātu kopu);
* Lietotājs labāk saprot kā rezultāts ir veidojies;
* Būs spēcīgs rīks profesionālu meklētāju rokās.

Negatīvie aspekti:

* Lietotājam ir jāzina formālā struktūra kuru konkrētā programma spēj interpretēt, respektīvi to kādā formā ir jābūt izteiksmei;
* Lietotājam ir jābūt zināšanām par to kā lietot loģiskos operatorus, līdz ar to arī noteiktā mērā rezultāta kvalitāte ir atkarīga no tā cik labi lietotājs spēs noformulēt savu vajadzību izmantojot vārdus un loģiskos operatorus;
* Spēj atgriezt rezultātus, kas atbilst lietotāja sniegtajai izteiksmei, bet nespēj rezultātus sakārtot pēc tā cik labi tie atbilst lietotāja informācijas vajadzībai, tātad spēj noteikt tikai to vai atbilst vai neatbilst;
* Bieži vien tiek atgriezti par daudz rezultāti vai arī par maz.

**Izteiksmes struktūra**

Lai izteiksme būtu darbspējīga tai ir jābūt noteiktā formā, tādā kas lietotājam būs viegli un intuitīvi uztverama un programmai būs interpretējama. Var izšķirt vismaz divus izteiksmes struktūru variantus - pamat variantu un paplašināto variantu, kur paplašinātā varianta sniegtās iespējas un to realizācija var atšķirties atkarībā no sistēmas vajadzībām un implementācijas.

### Pamatvariants

Pamat variantā izteiksme sastāv no vārdiem (simbolu virkne), kas savienoti ar loģiskajiem operatoriem. Vārdiem ir jābūt atdalītiem no loģiskajiem operatoriem ar vismaz vienu atstarpi, citādāk tos nevarētu viennozīmīgi atšķirt. Loģiskajiem operatoriem ir jābūt rakstītiem ar lielajiem burtiem, lai izteiksme būtu cilvēkam labāk lasāma un varētu intuitīvi nodalīt tekstu no loģiskā saikļa. Iekavas tiek lietotas lai izteiksmi padarītu skaidrāku, lasāmāku un lai noteiktu operāciju precedenci.

Formālā gramatika BNF formā definē augstāk aprakstīto boolean izteiksmes pamat varianta struktūru, kā arī nodrošina to, ka iekavas būs sabalansētas un veidos gan lietotājam, gan programmai saprotamu struktūru. Neterminālie simboli ir iekš <>, un terminālie ir tikai burti. -> apzīmē izveduma likumu, un | apzīmē izvedumu alternatīvas. <S> ir sākuma neterminālais simbols ar ko vienmēr uzsāk vārda ģenerēšanu. Gramatika tiek veidota speciāli priekš latviešu valodas:

<S> -> <OR> OR <AVAR> | <AND> AND <OVAR> | <NVAR> | NOT ( <AND> ) | NOT ( <OR> )

<OR> -> <OR> OR <AVAR> | <AVAR>

<AND> -> <AND> AND <OVAR> | <OVAR>

<AVAR> -> ( <AND> ) | NOT ( <AND> ) | <NVAR>

<OVAR> -> ( <OR> ) | NOT ( <OR> ) | <NVAR>

<NVAR> -> NOT <VAR> | <VAR>

<VAR> -> <UC><VAR> | <LC><VAR>

<UC> -> A | Ā | B | C | Č | D | E | Ē | F | G | Ģ | H | I | Ī | J | K | Ķ | L | Ļ | M | N | Ņ | O | P |

R | S | Š | T | U | Ū | V | Z | Ž

<LC> -> a | ā | b | c | č | d | e | ē | f | g | ģ | h | i | ī | j | k | l | ļ | m | n | ņ | o | p | r | s | š

| t | u | ū | v | z | ž

**Piemēri**

Lietotāja ievadīta izteiksme: “mežs AND avots AND NOT barība” (Atgriezīs visus tos dokumentus, kas satur vārdus mežs un avots, bet nesatur vārdu barība)

No kopīgā piemēra atgrieza: -

Lietotāja ievadīta izteiksme: “(mežs AND ekosistēma) OR (mežs AND avots)” (Atgriezīs visus tos dokumentus, kas satur vārdus mežs un ekosistēma, vai mežs un avots)

No kopīgā piemēra atgrieza: 1, 2, 4

Lietotāja ievadīta izteiksme: “NOT (mežs AND avots)” (Atgriezīs visus tos dokumentus, kas nesatur abus vārdus mežs un avots)

No kopīgā piemēra atgrieza: 2, 3, 4, 5

**Secinājumi**

Meklēšanai pēc šāda modeļa pietiktu ar bag of words datu struktūru, jo vārdiem tiek noteikta to eksistence vai ne eksistence dokumentos dažādās kombinācijās, nav nepieciešama nekāda papildus informācija tikai šī modeļa realizēšanai. Tomēr šis variants ir ierobežojošs, jo

tas paredz tikai atsevišķu vārdu sasaisti ar loģiskajiem operatoriem un nedod iespējas lietot vārdu pozīcijas.

### Paplašinātais variants

Par pamatu tiek ņemts pamat variants, bet tas tiek papildināts ar dažām papildus iespējām. Dod iespēju lietotājam meklēt pa frāzēm, frāze tiek likta pēdiņās. Atkarībā no implementācijas un konkrētās sistēmas vajadzībām šeit ir iespējas variēt ar dažādām meklēšanas papildiespējām un to apzīmējumiem. Paplašinātā boolean tipa meklēšana ievērojami paplašina lietotāja iespējas atrast precīzākus rezultātus. Apzīmējumi kas sevī ietver slīpsvītru tiek lietoti pa vidu diviem vārdiem vai frāzēm vai frāzes un vārda kombinācijas un attiecīgi arī attiecas uz reģionu starp šiem vārdiem vai frāzēm (iekļaujot vārdus vai frāzes). Ievades gramatika tiek papildināta ar noteiktām papildiespējām:

*3.2 tabula*

**Paplašinātā boolean modeļa apzīmējumi**

|  |  |
| --- | --- |
| Apzīmējums | Skaidrojums |
| ! | Jebkādi simboli un jebkādā daudzumā (wildcard). |
| /N | Noteikta frāze vai vārds iekš noteikta vārdu skaita, kur n ir vārdu skaits. |
| /s | Noteikta frāze vai vārds viena teikuma ietvaros. |
| /p | Noteikta frāze vai vārds viena paragrāfa ietvaros. |
| “N” | Frāze, kur N ir patvaļīgs vārdu skaits. |

**Piemēri**

Lietotāja ievadīta izteiksme: “ “mežs ir” /s avots “ (Atgriezīs visus tos dokumentus, kuros ir vismaz viens tāds teikums kurš sevī ietver frāzi “mežs ir” un pēc tās teikuma ietvaros seko vārds avots)

No kopīgā piemēra atgrieza: 1

Lietotāja ievadīta izteiksme: “ ekosistēma /p mežs “ (Atgriezīs visus tos dokumentus, kuros ir vismaz viens tāds paragrāfs kurš sevī ietver vārdu ekosistēma un pēc tās paragrāfa ietvaros seko vārds mežs)

No kopīgā piemēra atgrieza: 1

Lietotāja ievadīta izteiksme: “ mežs /5 sastāv “ (Atgriezīs visus tos dokumentus, kuros ir vismaz viena tāda frāze, kas 5 vārdu ietvaros satur vārdu mežs un pēc tā ir vārds sastāv)

No kopīgā piemēra atgrieza: 2

**Secinājumi**

Paplašinātais variants dod lietotājam daudz plašākas iespējas precīzi aprakstīt savu informācijas vajadzību salīdzinot ar pamat variantu. Toties tas prasa no lietotāja vairāk piedomāt un arī plašākas zināšanas par to kā veidot izteiksmes. Šim variantam labi derēs inverted index datu struktūra, bet lai varētu implementēt visas iespējas, datu struktūrā papildus vārdu pozīcijām svarīgi ir iekļaut informāciju par teikumu un paragrāfu robežām.

Secinājumi

Veicot darbu tika izpētītas un novērtētas dažādas datu struktūras, kuras var izmantot lai realizētu dokumentu meklēšanas sistēmu. Tika apskatīti algoritmi, kas nodrošina efektīvu meklēšanu laika ziņā un arī dažādi veidi kā lietotāja saskarnē var tikt veikta meklēšana. Katrai pieejai ir savi pozitīvie un negatīvie aspekti, kas darbā tiek skaidri aprakstīti. Salīdzinot darbā izpētīto metodiku, risinājuma veidus ar risinātās problēmas raksturojumu var veikt noteiktus secinājumus:

1. Veicot meklēšanu iepriekš realizētā datu struktūrā ir jāņem vērā fakts, ka meklēšanas rezultāts būs atkarīgs no šīs datu struktūras kvalitātes, ja tajā būs kļūdas, vai kādas neatbilstības, tad tādas arī būs rezultātā.
2. Svarīgi ir programmā implementēt efektīvu datu struktūru uzturēšanas mehānismu, jo lietotājam veicot izmaiņas dokumentos un pēc tam veicot meklēšanu var gadīties, ka vairs nevarēs atrast meklēto, datu struktūrai ir jābūt sinhronai ar aktuālo dokumentu kolekcijas stāvokli.
3. Noteikti ir vērts izmantot tādus meklēšanas modeļus, kā darbā aprakstītais “Patiesumvērtību meklēšanas modelis”, un it īpaši tā paplašināto variantu, šie modeļi sniedz izvērstākas meklēšanas iespējas, kas daudzās situācijās var labi palīdzēt atrast adatu siena kaudzē, bet tajā pašā programmā ir jābūt arī iespējai veikt meklēšanu tikai pēc frāzes neizmantojot papildus iespējas, jo ne vienmēr tās ir vajadzīgas.
4. Situācijās kur nepieciešams ir ātri atrast dokumentu kurš satur noteiktu informāciju šāda sistēma var ļoti noderēt, bet lai lietotājs spētu efektīvi lietot šādu sistēmu viņam ir jāzina dažas pamatlietas, kā piemēram ja viņa meklētais dokuments kādu iemeslu dēļ iepriekš nav ticis iekļauts indeksa datu struktūrā, tad sistēma to neatradīs (Šādām situācijām varētu paredzēt alternatīvu meklēšanas variantu, kur programma tieši lasa visus dokumentus no sākuma līdz beigām, bet šāda pieeja var prasīt ļoti daudz laika).
5. Lietotājs ne vienmēr savu informācijas vajadzību noformulēs tieši tā, kā tas ir dokumentā. Iespējams lietotājs uzrakstīs dažus vārdus citā locījumā, vai citā formā, vārdu glabāšana lemmas formā labi risina šo problēmu, bet arī paplašina atgriezto rezultātu kopu. Izmantojot paplašinātās meklēšanas iespējas, tai skaitā metadatus lietotājs var šo rezultātu kopu sašaurināt un ātrāk atrast nepieciešamo.

Izmantotās literatūras saraksts

Bag of words - Pieejams internetā: <https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428>

Bag of words - Pieejams internetā: https://www.educative.io/edpresso/what-is-the-bag-of-words-model

Bag of words - Pieejams internetā: https://machinelearningmastery.com/gentle-introduction-bag-words-model/

Inverted index - Pieejams internetā: https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html

Inverted index - Pieejams internetā: https://www.educative.io/edpresso/what-is-an-inverted-index

Stemming and lemmatization - Pieejams internetā: <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

Boolean retrieval - Pieejams internetā: <https://nlp.stanford.edu/IR-book/pdf/01bool.pdf>

Extended Boolean model - Pieejams internetā: <https://nlp.stanford.edu/IR-book/html/htmledition/the-extended-boolean-model-versus-ranked-retrieval-1.html>

Levenshtein Distance - Pieejams internetā: https://www.sciencedirect.com/topics/computer-science/levenshtein-distance

https://law.lclark.edu/live/files/9394-westlaw-terms-and-connectors-searching