თბილისის სახელმწიფო უნივერსიტეტი

შოთა ოდიშარია

N-გრამიანი მსგავსება და დისტანცია

კომპიუტერული მეცნიერება

მისანიჭებელი აკადემიური ხარისხი:

მაგისტრი

სამაგისტრო ნაშრომის ხელმძღვანელი:

მანანა ხაჩიძე, ტექნიკის მეცნიერებათა კანდიდატი, პროფესორი

თბილისი

2015 წელი

სარჩევი

ანოტაცია 3
შესავალი 5
1. უნიგრამული მსგავსება 7
1.1. სტანდარტული განმარტება 7
1.2. რეკურსიული განმარტება 8
1.3. დასაბუთებული ახსნა-განმარტება 8
1.4. დეკომპოზიციების შემცირებული რაოდენობა... 9
1.5. უნიგრამული მსგავსების მიღმა 10
2. N-გრამიანი მსგავსება 12
2.1. განმარტება 12
2.2. N-გრამიანი მსგავსების კომპილირება 13
2.3. დახვეწილი N-გრამიანი მსგავსება 14
3. N-გრამიანი დისტანცია 15
3.1. რეკურსიული განმარტება 15
3.2. განმარტება შემცირებული დეკომპოზიციების რიცხვით 16
3.3. N-გრამიანი დისტანციის ფორმულირება 16
3.4. N-გრამიანი დისტანციის ალტერნატიული ფორმულირება 16
3.5. N-გრამიანი დისტანციის სამი ფორმულირება... 16
4. სიტყვების N-გრამიანი მსგავსების საზომები 17
4.1. სიტყვების მსგავსების ნორმალიზაცია 17
4.2. სიმბოლოების გამყარება 18
დასკვნა 19
გამოყენებული ლიტერატურა 20
დანართი I: საერთო ფუნქციები - ალგორითმი, პროგრამული შესრულება 21
დანართი II: N-გრამიანი მსგავსება - ალგორითმი, პროგრამული შესრულება 24
დანართი III: N-გრამიანი დისტანცია - ალგორითმი, პროგრამული შესრულება 26
დანართი IV: N-გრამიანი დისტანციისა და მსგავსების მთვლელი პროგრამის ტესტები 28

ანოტაცია

დღეისათვის თითქმის ყველა პროგრამაში, რომელიც რაიმე სახის ტექსტური (და არა მარტო ტექსტური) მონაცემების დამუშავებაზეა დაფუძნებული, აუცილებელია ტექსტში ძებნის შესაძლებლობა. ზევრ პროგრამაში, მით უმეტეს, თუ პროგრამა გათვლილია ადამიანთან ინტერაქციაზე, მათი სპეციფიკიდან გამომდინარე, აუცილებელია მონაცემების მიახლობითი ძებნის არსებობა, რაც ზემოთ ხსენებულ ამოცანას კიდევ უფრო მეტად ართულებს.

მონაცემების მიახლოებითი ძებნის ერთ-ერთ მთავარ სირთულეს წარმოადგენს ის, რომ რთულია ალგორითმულად დადგინდეს და შეფასდეს რაიმე კრიტერიუმით ორი მოცემული სიტყვის მსგავსება, რამეთუ მსგავსება დამოკიდებულია კონკრეტული ადამიანის აღქმაზე, რისი ცალმხრივად განმარტება, ალბათ, შეუძლებელიც არის.

სიტყვების მსგავსების მთვლელი მრავალი ცნობილი და უკვე აპრობირებული ალგორითმი არსებობს. მათ შორის ყველაზე კარგად ცნობილი ალგორითმები დაფუძნებულია ფრაზების ცვლილების დისტანციასა (Edit Distance) და უდიდესი საერთო ქვე-ფრაზის სიგრძეზე (Longest common subsequence).

ნაშრომის მთავარი მიზანი ეფექტური ალგორითმის შემუშავება და პროგრამული უზრუნველყოფაა, ალგორითმის, რომელიც მოცემულ ორ ფრაზას შორის მსგავსებას დათვლის და მსგავსების კოეფიციენტს დაგვიბრუნებს, რაც შეიძლება დიდი სიზუსტით.

ნაშრომში წარმოდგენილია N-გრამიანი მსგავსების და დისტანციის ფორმალური და რეკურსიული განმარტებები. ნაჩვენებია, რომ ფრაზების ცვლილების დისტანცია და უდიდესი საერთო ქვე-ფრაზის სიგრძე კონკრეტული შემთხვევებია N-გრამიანი დისტანციისა და მსგავსებისა. ნაშრომში ასევე წარმოდგენილია N-გრამიანი მსგავსებისა და დისტანციის დათვლის ეფექტური ალგორითმები და მათი პროგრამული უზრუნველყოფა. ნაშრომში შემოღებულია ფრაზების მსგავსების ახალ საზომები, რომლებიც N-გრამიან ალგორითმებზე არის დაფუძნებული.

Abstract

Nowadays, almost in every program, that processes any kind of text data (and not only text data), it is almost a must to have a possibility of search in text. In many applications, especially in applications which interfare with humans, depanding on their specification, it's inevitable to have a service of approximate search. This, of course, more complicates the above said problem.

One of the main problems of the text approximate search is the fact, that it's very hard to determine and estimate the rate of two words' similarity. That's because, it depends on the human's interpretation and perception of the words. Thus, it's probably impossible to give the exact classification of the words' similarity.

However, there are a lot of known and already widely used algorythms that compute and calculate the similarity of phrases. Among them, probably the most known are best on the Edit Distance, also known as the Levinstains Distance and on the longest common subsequense (LCS).

The main goal of this paper is to define a proper algorithm for word similarity, which will inherit the pros of the already known algorithms and also defy their cons.

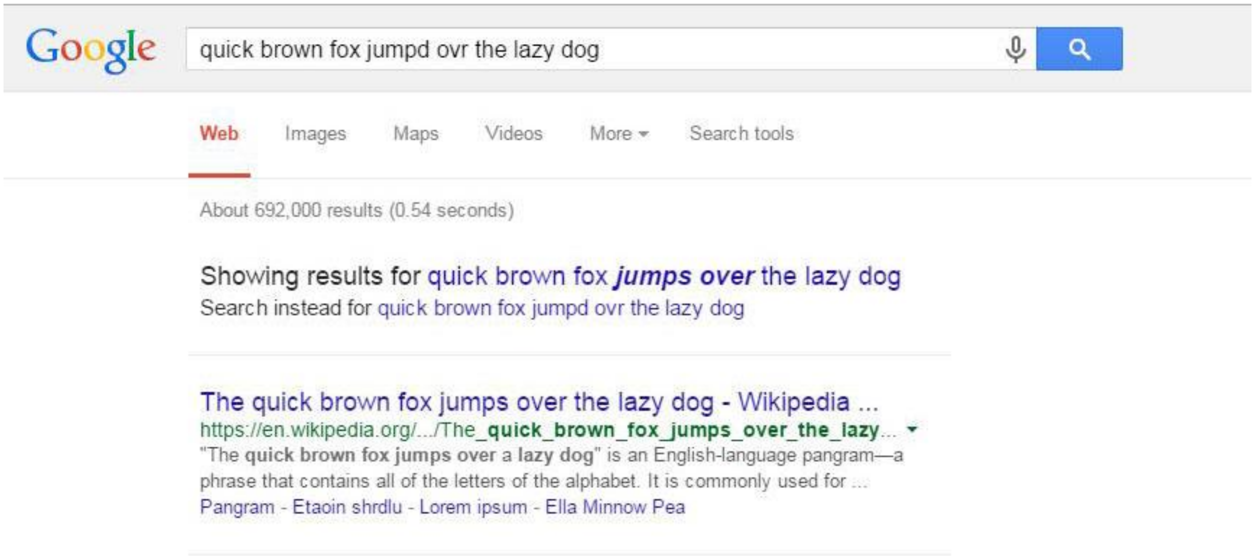
In the paper, we present the N-gramm similarity and distance algorithms, along with the code for computing them. We provide a formal and a recursive definitions. We show that the Edit distance and LCS are the specific cases of the N-gram similarity and distance. We provide new measurements for words similarity, which are based on the consept of N-gram similarity and distance.

შესავალი

დღეისათვის თითქმის ყველა პროგრამაში, რომელიც რაიმე სახის ტექსტური (და არა მარტო ტექსტური) მონაცემების დამუშავებაზეა დაფუძნებული, აუცილებელია ტექსტში ძებნის შესაძლებლობა. ბევრ პროგრამაში, მით უმეტეს, თუ პროგრამა გათვლილია ადამიანთან ინტერაქციაზე, მათი სპეციფიკიდან გამომდინარე, აუცილებელია მონაცემების მიახლობითი ძებნის არსებობა, რაც ზემოთ ხსენებულ ამოცანას კიდევ უფრო მეტად ართულებს.

მონაცემების მიახლოებითი ძებნის ერთ-ერთ მთავარ სირთულეს წარმოადგენს ის, რომ რთულია ალგორითმულად დადგინდეს და შეფასდეს რაიმე კრიტერიუმით ორი მოცემული სიტყვის მსგავსება, რამეთუ მსგავსება დამოკიდებულია კონკრეტული ადამიანის აღქმაზე, და ამ აღქმის ცალმხრივად განმარტება შეუძლებელიც არის.

მიახლოებითი ძიების ერთ-ერთ, ალბათ, ყველაზე ნათელ მაგალითს წარმოადგენს ძიების ძრავი Google. Google-ს საძიებო სისტემაში ფრაზის ჩაწერისას პროგრამა ეძებს თქვენს შეყვანილ ტექსტს მონაცემთა ბაზაში, რომ თქვენთვის სასურველი რეზულტატი გამოიტანოს. იმის გამო, რომ მომხმარებელი ადამინია, და დიდია ალბათობა იმისა, რომ ტექსტის შეყვანისას ადამინამა შესაძლებელია შეცდომა დაუშვას, ან თუნდაც, არ იცოდეს ზუსტად საძიებო ტექსტი, ამიტომ Google ამუშავებს თქვენს შეყვანილ ფრაზას და თავის მონაცემთა ბაზაზე დაყრდნობით ცვლის მას თუ ამის საჭიროებას ხედავს (მაგალითად თუ თქვენი ტექსტი ზუსტად ვერ მოიძებნა ან მოიძებნა ბევრი ისეთი ტექსტი, რომელიც ძალიან გავს თქვენს შეყვანილ ფრაზას, მაგრამ ზუსტად ანალოგიური არ არის) (სურათი 1).



სურათი 1: სურათზე ნაჩვენებია Google-ს Auto-Corrector-ი, რომელიც საძიებო ტექსტს *„quick brown fox jumpd ovr the lazy dog“* ავტომატურად ცვლის გამართული წინადადებით - *„quick brown fox jumped over the lazy dog“*.

როგორც უკვე აღვნიშნეთ, მიახლოებითი ძებნის ამოცანის ერთ-ერთ ყველაზე დიდ სირთულეს სიტყვებს შორის მსგავსების დადგენა და შეფასება წარმოადგენს. სწორედ ეს კონკრეტულ ამოცანა არის განხილული ნაშრომში. ფრაზებს შორის მსგავსების ამოცანა, არამხოლოდ მიახლოებით ძებნაში, არამედ სხვა მრავალ არანაკლებ მნიშვნელოვან პროგრამებსა და დარგებშია აქტუალური. ასეთია, მაგალითად, მედიცინა, სადაც მრავალი დასახელების მედიკამენტები არსებობს, და ამ მედიკამენტებს შორის ასევე მრავალია ისეთები, რომელთაც მსგავსი დასახელებები აქვთ. ამ შემთხვევაში კარგი იქნება ისეთი ხელსაწყოს ქონა, რომელიც შეაფასებს დასახელებების მსგავსებას, თუნდაც იმისთვის, რომ არ მოხდეს მედიკამენტების ერთმანეთში არევა. მაგალითად, გამოსადეგია იმის ცოდნა, რომ მედიკამენტები *ტორადოლი* და *ტეგრედოლი,* უფრო მსგავს დასახელებიანი მედიკამენტებია, ვიდრე, ვთქვათ, *ტორადოლი* და *ინდერალი.* გარდა მედიცინისა, ფრაზების მსგავსების დადგენა / შეფასების ამოცანა ასევე გამოსადეგია ბიოინფორმატიკაში, საუბრის ამოცნობის პროგრამებში, კომპიუტერულ თარგმანში, ლექსიკოგრაფიაში, დიალექტოლოგიაში [9] და ა.შ.

სიტყვების მსგავსების მრავალი უკვე ცნობილი და პრაქტიკაში აპრობირებული საზომი არსებობს. მათ შორის საკმაოდ ცნობილია *ცვლილების დისტანცია* (EDIT), ასევე ცნობილი როგორც *ლევინშტეინის-დისტანცია* რომელიც განსაზღვრულია ფრაზაში ელემენტარული ცვლილებების მინიმალური რაოდენობით. ელემენტარულ ცვლილებაში იგულისხმება ცვლილება (*replace*), წაშლა (*delete*) ან დამატება (*add*).

ამ ამოცანის გადასაწყვეტად ასევე ცნობილი მიდგომაა - ორი სიტყვისთვის უდიდესი საერთო ქვე-ფრაზის ზომის პოვნა (longest common subsequence LCS) და ამ უდიდესი ქვე-ფრაზის ზომით სიტყვების მსგავსების შეფასება. LCS პირდაპირ კავშირშია ლევინშტეინის დისტანციასთან. სხვა მსგავსების საზომები ძირითადად დამოკიდებულია ფრაზებში საერთო q-გრამიანი (ქვე-ფრაზა, რომელიც შედგება q სიმბოლოსგან) ქვე-ფრაზების რაოდენობაზე.

ჩვენ წარმოვადგენთ N-გრამიანი მსგავსების და დისტანციის ფორმალურ და რეკურსიულ განმარტებებს. ასევე წარმოვადგენთ ეფექტურ ალგორითმებს და მათ პროგრამულ უზრუნველყოფას. ჩვენ შემოვიღებთ ფრაზების მსგავსების ახალ საზომებს, რომლებიც N-გრამიან ალგორითმებზე იქნება დაფუძნებული.

ჩვენ დავიწყებთ N-გრამიან მსგავსებით, რამეთუ ჩვენი აზრით ის კონცეპტუალურად უფრო მარტივია N-გრამიან დისტანციასთან შედარებით.

1. უნიგრამული მსგავსება

ამ სექციაში ჩვენ განვიხილავთ LCS (Longest Common Substring - უდიდესი საერთო ქვე-ფრაზა) ზომის ცნებას, რომელსაც ფრაზების მსგავსების შეფასების კონტექსტში *უნიგრამულ მსგავსებას* ვუწოდებთ. მას შემდეგ, რაც განვმარტავთ უდიდესი საერთო ქვე-ფრაზის ზომის პოვნის ამოცანას სტანდარტულად, ჩამოვაყალიბებთ ალტერნატიულ, მაგრამ ტოლფას ფორმულირებას LCS ზომისა. ასევე ჩამოვაყალიბებთ რეკურსიულ განმარტებას და დავინახავთ, რომ LCS და ლევინშტეინის დისტანცია პირდაპირ კავშირშია ერთმანეთთან. და საბოლოოდ დავინახავთ, რომ LCS-ს რეკურსიული ფორმულის განზოგადებით ის ბუნებრივად განივრცობა N-გრამიან მსგავსებასა და დისტანციაზე (N-გრამიანი მსგავსება და დისტანცია არ არის იგივე ცნება რაც ცნობილი q-გრამიანი მსგავსება და დისტანცია, რომელიც თავის თავში გულისხმობს ორ ფრაზაში უბრალოდ საერთო q-გრამიანი ქვეფრაზების რაოდენობის დათვლას, N-გრამიან მსგავსება / დისტანციას კი განვმარტავთ მე-2 და მე-3 სექციებში).

1.1. სტანდარტული განმარტება

LCS -ის (Longest Common Substring - უდიდესი ქვე-ფრაზა) სტანდარტული განმარტება მდგომარეობს შემდეგში [3]: მოცემული $X = x_1, \dots, x_k$ მიმდევრობისთვის, მეორე $Z = z_1, \dots, z_m$ მიმდევრობა არის X-ის ქვე-მიმდევრობა თუ არსებობს X მიმდევრობის ინდექსების მკაცრად ზრდადი $i_1 \dots i_m$ მიმდევრობა, ისეთი, რომ ყოველი $j = 1, \dots, m$ -სთვის ჩვენ გვაქვს $x_{i_j} = z_j$. მაგალითად, 'ლით' არის 'მაგალითი'-ის ქვე-მიმდევრობა.

მოცემული ორი X და Y მიმდევრობებისთვის, ჩვენ ვამბობთ, რომ Z მიმდევრობა ამ მიმდევრობების საერთო ქვე-მიმდევრობაა, თუ Z ქვე-მიმდევრობაა X-ის და ასევე ქვე-მიმდევრობაა Y-ის.

LCS ამოცანაში, ჩვენ მოცემული გვაქვს ორი მიმდევრობა (ორი სიტყვა) და გვსურს მათ ქვემიმდევრობებს (ქვე-ფრაზებს) შორის უდიდედსის პოვნა. მაგალითად, LCS ორი ფრაზისთვის „კომპიუტერი“ და „იუპიტერი“ არის „იუპტერი“.

ჩვენი ძირითადი ამოცანისთვის, რომელიც მდგომარეობს ორ სიტყვას შორის მსგავსების დადგენაში, უდიდესი ქვე-ფრაზის მხოლოდ *ზომა* არის მნიშვნელოვანი. თავად LCS ქვე-ფრაზის მნიშვნელობა ამ ამოცანისთვის არ არის მნიშვნელოვანი. LCS-ის ზომის გამოთვლა, როგორც ორი *string* ტიპის ცვლადის ფუნქცია, ცალკე საინტერესო ამოცანაა [2].

1.2. რეკურსიული განმარტება

ჩვენ წარმოვადგენთ შემდეგი $s(X, Y)$ ფუნქციის (რომელიც X-ის და Y-ის LCS-ის ზომის ექვივალენტურია), ფორმალურ, რეკურსიულ განმარტებას.

ვთქვათ, $X = x_1, \dots, x_k$ და $Y = y_1, \dots, y_l$ არიან k და l ზომის ფრაზები, რომლებიც სასრულ-ალფაბეტიანი სიმბოლოებისაგან შედგებიან. ფორმულირების გასამარტივებლად, ჩვენ წარმოვადგენთ შემდეგ ჩანაწერს (Smyth [10]).

დავუშვათ, რომ $\Gamma_{i,j} = (x_1 \dots x_i, y_1 \dots y_j)$ არის X-ის და Y-ის პრეფიქსების წყვილი, ხოლო $\Gamma^*_{i,j} = (x_{i+1} \dots x_k, y_{j+1} \dots y_l)$ არის X-ის და Y-ის სუფიქსების წყვილი.

მიმდევრობებისთვის, რომელთა ზომა არის ერთი ან ნაკლები ჩვენ ვმარტავთ s-ს პირდაპირ:

$$s(x, e) = 0, \quad s(e, y) = 0, \quad s(x, y) = \begin{cases} 1 \text{ თუ } x = y \\ 0 \text{ სხვა შემთხვევაში} \end{cases}$$

სადაც e ცარიელი მიმდევრობაა, ხოლო x და y წარმოადგენენ თითო სიმბოლოს.

ერთზე მეტ სიმბოლოიანი მიმდევრობებისთვის ჩვენ ვმარტავთ s-ს რეკურსიულად:

$$s(X, Y) = s(\Gamma_{k,l}) = \max_{i,j}(s(\Gamma_{i,j}) + s(\Gamma^*_{i,j}))$$

i და j-ს მნიშვნელობები ზემოთ მოყვანილ ფორმულაში შეზღუდულია იმ პირობით, რომ ორივე $\Gamma_{i,j}$ და $\Gamma^*_{i,j}$ არანულოვანი ქვე-ფრაზებია. უფრო სპეციფიურად კი i და j-ს მნიშვნელობები მოცემულია შემდეგი წყვილების ერთობლიობიდან:

$$D(k, l) = \{0, \dots, k\} X \{0, \dots, l\} - \{(0,0), (k, l)\}$$

მაგალითად, $D(2,1) = \{(0,1), (1,0), (1,1), (2,0)\}$

ინდუქციის მეთოდით მარტივად შეიძლება ჩვენება, რომ $s(X, Y)$ ყოველთვის X, Y ფრაზების უდიდესი ქვე-ფრაზის ზომის ტოლია.

1.3. დასაბუთებული ახსნა-განმარტება

ჩვენი რეკურსიული განმარტება იყენებს LCS-ის ნაწილობრივ კომპოზიციურობას. რა თქმა უნდა, LCS არ არის კომპოზიციური, იმ გაგებით, რომ კონკატენირებული ფრაზების უდიდესი ქვეფრაზის ზომა არ არის აუცილებელი, რომ ამ ფრაზების დეკომპოზიციური ფრაზების LCS-ის ტოლი იყოს. ანუ, ვთქვათ, *LSC(ab,a)=1* და *LCS(c,bc)=1,* მაგრამ *LCS(abc,abc)=3.* ანუ LCS არ არის კომპოზიციური, მაგრამ ნათელია, რომ კონკატენირებული ფრაზების უდიდესი ქვეფრაზის ზომა ყოველთვის მეტია ან ტოლია მისი შემადგენელი ფრაზების LCS-ის ზომათა ჯამის, ანუ:

$$S(X_1, Y_1) + S(X_2, Y_2) \leq S(X_1 + X_2, Y_1 + Y_2)$$

ანუ შეგვიძლია ვთქვათ, რომ $S(X, Y)$ არის სუპერადიტიური. ყოველთვის შესაძლებელია შევადაროთ ორი ფრაზის LCS მათი ქვე-ფრაზების LCS-ების შედარებით, იმის გათვალისწინებით, რომ ფრაზის ქვე-ფრაზებად დეკომპოზიცია ინარჩუნებს მთავარი ფრაზის იდენტურობას.

1.4. დეკომპოზიციების შემცირებული რაოდენობა

D წყვილების ერთობლიობა მოიცავს წყვილი სიტყვების ყველა შესაძლო დეკომპოზიციას. ყველა შესაძლო დეკომპოზიციის რიცხვი წყვილი სიტყვებისთვის (k და *1 სიგრძის*) არის *(k+1)*(l+1)-2.*

ჩვენ შეგვიძლია დეკომპოზიციების სიმრავლის შემცირება ფუნქცია s ის მნიშვნელობის შეუცვლელად. ვთქვად D' არის დეკომპოზიციების შემდეგი სიმრავლე:

$$D'(k, l) = \{k - 1, l\} \times \{l - 1, l\} - \{(0,0), (k, l)\}$$

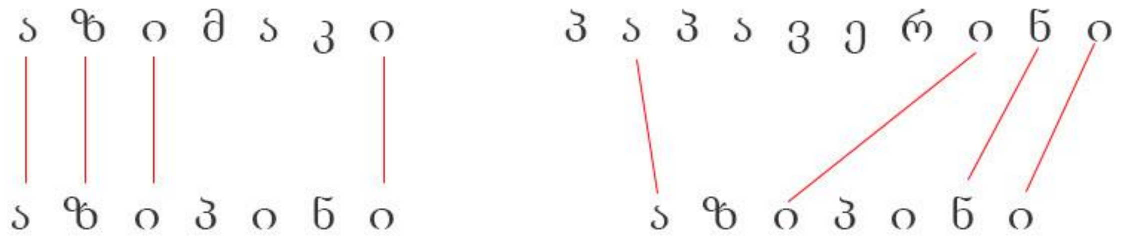
მაგალითად, $D'(2,1) = \{(1,0), (1,1), (2,0)\}$. D' არასოდეს შეიცავს 3-ზე მეტ დეკომპოზიციას. თუ D-ს ჩავანაცვლებთ D'-ით ჩვენს რეკურსიულ განმარტებაში (სექცია 1.2), მივიღებთ ალტერნატიულ, ექვივალენტურ ფორმულირებას s ფუნქციისა:

$$s(X, Y) = s(\Gamma_{k,l}) = max(s(\Gamma_{k-1,l}), s(\Gamma_{k,l-1}), s(\Gamma_{k-1,l-1}) + s(\, x_k, x_l))$$

LCS სიგრძის ალტერნატიული ფორმულირების გამოთვლა კარგად-ცნობილი დინამიური პროგრამირების ალგორითმის გამოყენებით არის შესაძლებელი [14].

1.5. უნიგრამული მსგავსების მიღმა

უნიგრამული მსგავსების მთავარი სისუსტე მის კონტექსტისადმი არასენსიტიურობაშია. მაგალითისთვის, შეგვიძლია მოვიყვანოთ სიტყვების წყვილები (სურათი 2), რომლებსაც ერთნაერი უნიგრამული მსგავსების კოეფიციენტი აქვთ, მაგრამ აშკარად სხვადასხვა მსგავსების დონე, ასეთებია მაგალიტად მედიკამენტების დასახელებები: „აზიმაკი“ - „აზიპინი“ და „პაპავერინი“ – „აზიპინი“, როგორც ვხედავთ პრეპარატის სახელი „აზიპინი“ ორივე სხვა სახელთან უდიდესი საერთო ქვესიმრავლის სიგრძე აქვს 4 სიმბოლო, როდესაც ნათელია, რომ მსგავსება ბევრად უფრო ახლოსაა აზიმაკისა და აზიპინის ვიდრე მეორე წყვილის.



სურათი 2: *სიტყვების ორი წყვილი ერთნაერი საერთო ქვე-სიმრავლის სიგრძით, მაგრამ განსხვავებული მსგავსებით.*

ამ მაგალითიდან კიდევ შეგვიძლია დავასკვნათ, რომ მეზობელი სიმბოლოების ტოლობა ბევრად უფრო ძლიერი შეფასებაა მსგავსების დონის, ვიდრე არა მეზობელი სიმბოლოების ტოლობა. ასევე შეგვიძლია ვთქვათ, რომ პარალელური სიმბოლოების ტოლობის დროს (სიმბოლოები რომლებიც ერთნაერად არიან დაშორებული ერთმანეთისგან) სიტყვები უფრო მსგავსია, ვიდრე არაპარალელური სიმბოლოების ტოლობისას.

მსგავსების საზომი ერთეულების ერთობლიობა, რომლებიც კონტექსტს ითვალისწინებენ დაფუძნებულია ე.წ. კამათლის კოეფიციენტზე *(Dice coefficient [1]).* საზომი ერთეულები განსაზღვრულია ფრაზების საერთო n-გრამიანი ქვეფრაზების რაოდენობითა და ფრაზებში n-გრამების ჯამური რაოდენობით შემდეგ ნაერად:

$$\frac{2 \times |n - grams(X) \cap n - grams(Y)|}{|n - grams(X)| + |n - grams(Y)|}$$

სადაც *n-grams(X)* არის *X*-ის ყველა *n*-გრამიანი (სიმბოლოიანი) ქვე-ფრაზებისაგან შემდგარი სიმრავლე. კამათლის კოეფიციენტი ბიგრამებით (DICE) არის ერთ-ერთი ყველაზე

პოპულარული მეთოდი ფრაზების მსგავსების შესაფასებლად. მაგალითისთვის

$$DICE(Zantac, Contac) = \frac{(2*3)}{5+5} = 0.6$$

რადგან სამი ბიგრამი არის საერთო.

მიუხედავად იმისა, რომ LCS-თან შედარებით უფრო სენსიტიურია კონტექსტთან, DICE შეფასებას მაინც აქვს თავისი პრობლემები.

პირველ რიგში, „მცირე რეზოლუციის“ DICE ხშირად საერთოდ ვერანაერ მსგავსებას ვერ პოულობს სტრინგებში, რომლებიც ძალზედ გვანან ერთმანეთს, მაგალითისთვის: *Verelan/Virilon* ამ სიტყვების DICE კოეფიციენტი იქნება 0, რადგან ამ ორ სიტყვას არცერთი ბიგრამი არ აქვთ საერთო.

მეორე პრობლემა იმაში მდგომარეობს, რომ DICE კოეფიციენტმა შესაძლებელია დააბრუნოს უმაღლესი მსგავსების კოეფიციენტი 1, მაგრამ ფრაზები არ იყოს ერთმანეთის ტოლი. ასეთი მაგალითად: *Xanex* და *Nexan* - ამ ორი პრეპარატის სახელის DICE მსგავსების კოეფიციენტი არის 1, რადგან ორივე ერთიდაიგივე ბიგრამებისგან შედგება: *{xa, an, ne, ex},* მაგრამ მიუხედავად ამისა, სახელები სხვადსახვაა.

კიდევ პრობლემა იმაში მდგომარეობს, რომ DICE ხშირად ისეთ n-გრამიან ფრაზებს აკავშირებს ერთმანეთს, რომლებიც სტრინგის სხვადასხვა ადგილას ზის, რასაც წესით მსგავსებისთვის არ უნდა ითვალისწინებდეს, ანუ მაგალითისთვის, *V**ol**taren* და *Tramad**ol*** სახელების *ol* ბიგრამი არ უნდა მატებდეს ფრაზების მსგავსების კოეფიციენტს.

ბრიუ და მაკ’კელვიმ (Brew and McKelvie [1]) წარმოადგინეს DICE კოეფიციენტის განვრცობილი ვარიანტი, სახელად XXDICE, რომელშიც მსგავსი n-გრამიანი ქვე-ფრაზების მსგავსობის წონა საერთო ჯამურ მსგავსების შედეგში დამოკიდებულია მათ აბსოლუტურ მდებარეობაზე ფრაზაში. XXDICE-მ საუკეთესო შედეგი აჩვენა ბევრ სხვა მსგავსების კოეფიციენტებთან შედარებით, და ასევე ხშირად არის გამოყენებული სხვა მკვლევარების მიერ (როგორიც არის [11]).

სამწუხაროდ, XXDICE-ს განმარტება დეფიციტურია, და არ აკონკრეტებს, რომელი ბიგრამი უნდა იყოს არჩეული და კალკულაციაში გათვალისწინებული, როდესაც ბიგრამები უნიკალური არ არის. არსებობს რამდენიმე გზა განმარტების დასასრულებლად, მაგრამ ამ შემტხვევაში ეს შეფასება ხდება იმპლიმენტაციაზე დამოკიდებული, რაც იმას ნიშნავს, რომ შედეგები ვეღარ იქნება სრულად რეპროდუცირებადი.

XXDICE-ს მაგალითი გვაჩვენებს, რომ აუცილებელია ფრაზების მსგავსების შეფასებების მკაცრი განსაზღვრება.

შემდეგ სექციაში ჩვენ წარმოვადგენთ N-გრამიანი მსგავსების ცნებას, რომლის მიზანიც არის, რომ შეაჯამოს LCS სიგრძისა და კამათლის კოეფიციენტის დადებითი მხარეები და ასევე გამორიცხოს მათი უარყოფითი მხარეები.

2. N-გრამიანი მსგავსება

მთავარი იდეა N-გრამიანი მსგავსების არის უდიდესი ქვე-ფრაზის ამოცანის განზოგადოება ისე, რომ ქვეფრაზები შედგებოდნენ N-სიმბოლოსგან, და არ იყვნენ მხოლოდ თითო სიმბოლოსგან შემდგარი, როგორც ეს უნიგრამულ მსგავსებაში იყო. ჩვენ გავაკეთებთ N-გრამიანი მსგავსების ფორმულირებას როგორც ფუნქცია Sn, სადაც n ფიქსირებული პარამეტრია. s_1 ფუნქცია უნიგრამული მსგავსების ფუნქციის ექვივალენტურია, რომელიც 2.2. სექციაში აღვწერეთ.

2.1. განმარტება

იმისათვის, რომ რეკურსიულად განვმარტოთ N-გრამიანი მსგავსება, ჩვენ მცირედით შევცვლით ჩვენს განმარტებას Γ-სთან მიმართებაში. როდესაც ჩვენ საქმე გვაქვს n-გრამთან $n > 1$ -სთვის, ჩვენ მოვითხოვთ, რომ $\Gamma_{i,j}$ და $\Gamma^*{}_{i,j}$ აუცილებლად უნდა შეიცავდნენ ერთ სრულ n-გრამს მაინც. ეს მოთხოვნა ძალაში რჩება ჩვენს წინა განმარტებაშიც $n=1$ უნიგრამულ მსგავსებაში. თუ ორივე ფრაზა ნაკლებია n-ზე, მაშინ s_n განუსაზღვრელია.

ყველაზე მარტივ ვარიანტში, როდესაც მხოლოდ ერთი სრული n-გრამი გვაქვს ერთ-ერთ სიტყვაში, მაშინ N-გრამიანი მსგავსება განიმარტება შემდეგნაერად:

$$s_n(\Gamma_{k,l}) = 0 \text{ თუ } (k = n \wedge \ l < n) \vee (l = n \wedge \ k < n)$$

ვთვათ $\Gamma_{i,j}^n = (x_{i+1} \dots x_{i+n}, y_{i+1} \dots y_{i+n})$ არის n-გრამების (n სიმბოლოიანი ფრაზების) წყვილი X-ში და Y-ში. თუ ორივე ფრაზა შეიცავს ზუსტად ერთ ცალ n-გრამს, მაშინ ჩვენი განმარტება n-გრამიანი მსგავსებისა არის მკაცრად ბინარული: s_n არის 1, თუ n-გრამები იდენტურია, და არის 0 თუ n-გრამები განსხვავდება. (მოგვიანებით, ჩვენ განვიხილავთ ამ განმარტების განვრცობის შესაძლებლობებს.)

$$s_n(\Gamma_{n,n}) = s_n(\Gamma_{0,0}^n) = \begin{cases} 1 \text{ თუ } \forall\ 1 \le u \le n\ x_u = y_u \\ 0 \text{ სხვა შემთხვევაში} \end{cases}$$

უფრო გრძელი მიმდევრობებისთვის, კი ჩვენ განვმარტავთ N-გრამიან მსგავსებას რეკურსიულად:

$$s(X,Y) = s_n\left(\Gamma_{k,l}\right) = \max_{i,j}(s_n\left(\Gamma_{i+n-1,j+n-1}\right) + s_n(\Gamma^*_{i,j}))$$

i და j ცვლადების მნიშვნელობები ზემოთ მოცემულ ფორმულაში შეზღუდულია ისე, რომ ორივე $\Gamma_{i,j}$ და $\Gamma^*_{i,j}$ აუცილებლად უნდა შეიცავდეს ერთ n-გრამს მაინც. უფრო სპეციფიურად კი, i და j-ს დასაშვები მნიშვნელობები მოცემულია $D(k-n+1, l-n+1)$ წყვილებით, სადაც D არის სიმრავლე განმარტებული 2.2 სექციაში.

2.2. N-გრამიანი მსგავსების კომპილირება

ისევე, როგორც უნიგრამული მსგავსების რეკურსიული ალგორითმის დროს, N-გრამიანი მსგავსების დროსაც დეკომპოზიციების რიცხვის სამამდე შემცირება საკმარისია და კომპილაციისთვის მიზანშეწონილიც.

ანუ შემცირებული დეკომპოზიციების რიცხვით ჩვენი n-გრამიანი მსგავსების ფორმულირება მიირებს შემდეგ სახეს:

$$s_n\left(\Gamma_{k,l}\right) = \max\left(s_n\left(\Gamma_{k-1,l}\right), s_n\left(\Gamma_{k,l-1}\right), s_n\left(\Gamma_{k-1,l-1}\right) + s_n\left({\Gamma^n}_{k-n,l-n}\right)\right)$$

როგორც ვხედავთ, n=1-სთვის ეს ფორმულირება იღებს უკვე განმარტებული უნიგრამული მსგავსების რეკურსიული ფორმულირების სახეს (LSC 1.4.)

ამ ფორმულირების ალგორითმი და პროგრამული უზრუნველყოფა განხილულია კომენტარებით ნაშრომის ბოლოს დანართ I, დანართ II-სა და დანართ III-ში.

2.3. დახვეწილი N-გრამიანი მსგავსება

მიუხედავად იმისა, რომ ზემოთ განმარტებული ბინარული n-გრამიანი მსგავსება საკმაოდ ეფექტურია სიტყვების მსგავსების საპოვნელად, მას მაინც აქვს გარკვეული პრობლემები, რომლებიც გამოიხატება შემდეგში: ჩვენს მიერ აღწერილი n-გრამიანი მსგავსება არ ასხვავებს n-გრამებს, რომლებიც ძალიან მცირედით განსხვავდებიან და n-გრამებს, რომლებიც აბსოლუტურად განსხვავდებიან. ასე მაგალითად, თუ გვაქვს ორი მედიკამეტის დასახელება - *verelan* და *virilon (სურათი 3)*, და ვიყენებთ n-გრამიან მსგავსებას n=2-ისთვის, ანუ ბიგრამულ მსგასვებას, მაშინ ამ ორ სახელში, იმის გამო, რომ არცერთი საერთო ბიგრამი არ გააჩნიათ მათ, მსგავსება ნულის ტოლი იქნება, მაშინ როდესაც ბიგრამები მართალია განსხვავდებიან, მაგრამ ძალიან გვანან ერთმანეთს.

ასეთ შემთხვევაში ჩვენ განვიცილავთ ორ შესაძლო დახვეწას n-გრამიანი მსგავსებისა: *ყოველმხრივ მიდგომას* და *პოზიციურ* მიდგომას.

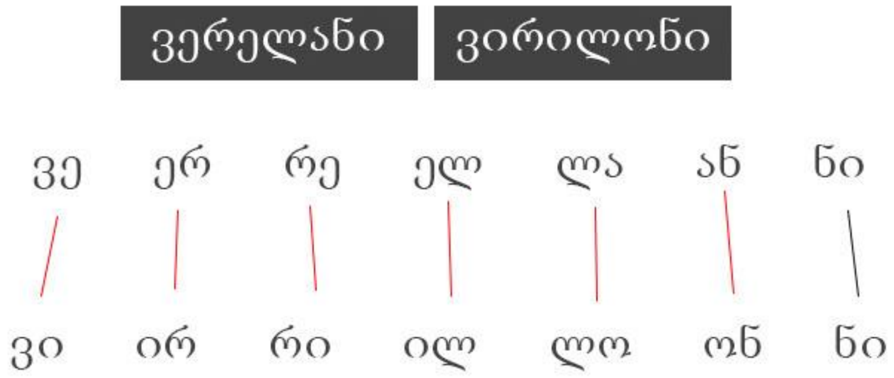
ყოველმხრივი მიდგომა მდგომარეობს შემდეგში - ფრაზებს შორის მსგავსებას ვითვლით ფრაზების n-გრამებში სტანდარტულ უნიგრამულ მსგავსების დათვლით:

$$s_n\left(\Gamma_{i,j}^n\right) = \frac{1}{n} s_1(\Gamma_{i,j}^n)$$

მეორე ალტერნატივა არის, როგორც უკვე ვთქვით, პოზიციური მიდგომა, რომლის დროსაც უბრალოდ ვითვლით იდენტურ უნიგრამებს ფრაზების n-გრამების შესაბამის პოზიციებზე:

$$s_n\left(\Gamma_{i,j}^n\right) = \frac{1}{n} \sum_{u=1}^{n} s_1(x_{i+u}, y_{i+u})$$

პოზიციური მიდგომის დადებითი მხარე, რა თქმა უნდა, არის ის, რომ ის უფრო სწრაფად შესაძლებელია დაკომპილირდეს, ვიდრე ყოველმხრივი მიდგომა. ასევე აღსანიშნავია ისიც, რომ ყოველმხრივი მიდგომა n-გრამიანი მსგავსების დათვლისას უფრო ზუსტად ითვლის ფრაზებს შორის მსგავსებას.



სურათი 3: მსგავსება სიტყვებს შორის „ვერელანი“ და „ვირილონი“. შავი ხაზით გამოსახულია იდენტური ბიგრამები, ხოლო წითელი ხაზებით გამოსახულია მსგავსი ბიგრამები.

3. N-გრამიანი დისტანცია

რამეთუ სტანდარტული რედაქტირების (EDIT) დისტანცია, იგივე ლევინშტეინის დისტანცია თითქმის LCS - უდიდესი საერთო ქვე-ფრაზის ზომის ორმაგი ცნებაა, n-გრამიანი დისტანციის განმარტებაც n-გრამიან მსგავსებისგან მხოლოდ დეტალებში განსხვავდება:

3.1. რედაქტირების დისტანციის რეკურსიული განმარტება

$$d(x, e) = 1,$$

$$d(e, y) = 1,$$

$$d(x, y) = \begin{cases} 0 \; \textit{თუ}\; x = y \\ 1 \; \textit{სხვა შემთხვევაში} \end{cases}$$

$$d(X, Y) = d(\Gamma_{k,l}) = \min_{i,j}\left(d(\Gamma_{i,j}) + d(\Gamma^{*}_{i,j})\right)$$

3.2. რედაქტირების დისტანციის ალტერნატიული ფორმულირება (დეკომპოზიციების შემცირებული რიცხვით)

$$d(X,Y) = d(\Gamma_{k,l}) = \min\left(d(\Gamma_{k-1,l}) + 1, d(\Gamma_{k,l-1}) + 1, d(\Gamma_{k-1,l-1}) + d(x_k, y_k)\right)$$

3.3. N-გრამიანი დისტანციის ფორმულირება

$$d_n(\Gamma_{k,l}) = 1 \text{ თუ } (k = n \wedge \ l < n) \vee (l = n \wedge \ k < n)$$

$$d_n(\Gamma_{n,n}) = d_n(\Gamma^n_{0,0}) = \begin{cases} 0 \text{ თუ } \forall\ 1 \leq u \leq n \ x_u = y_u \\ 1 \text{ სხვა შემთხვევაში} \end{cases}$$

$$d(X,Y) = d_n(\Gamma_{k,l}) = \min_{i,j}(d_n(\Gamma_{i+n-1,j+n-1}) + d_n(\Gamma^*_{i,j}))$$

3.4. N-გრამიანი დისტანციის ალტერნატიული ფორმულირება (დეკომპოზიციების შემცირებული რიცხვით)

$$d_n(\Gamma_{k,l}) = \min\left(d_n(\Gamma_{k-1,l}) + 1, d_n(\Gamma_{k,l-1}) + 1, d_n(\Gamma_{k-1,l-1}) + d_n({\Gamma^n}_{k-n,l-n})\right)$$

3.5. N-გრამიანი დისტანციის ფორმულირების სამი ვარიანტი $\boldsymbol{d_n({\Gamma^n}_{i,j})}$

- ბინარული n-გრამიანი დისტანცია, განმარტებული (3)-ში
- ყოველმხრივი n-გრამიანი დისტანცია: $d_n\left(\Gamma^n{}_{i,j}\right) = \frac{1}{n} d_1\left(\Gamma^n{}_{i,j}\right)$
- პოზიციური n-გრამიანი დისტანცია: $d_n\left(\Gamma^n{}_{i,j}\right) = \frac{1}{n} \sum_{u=1}^{n} d_1\left(x_{i+u}, y_{j+u}\right)$

პოზიციური n-გრამიანი დისტანცია ექვივალენტურია ყოველმხრივი n-გრამიანი დისტანციის n=2-სთვის. სამივე ფორმულირება ექვივალენტურია n=1-სთვის.

4. სიტყვების N-გრამიანი მსგავსების საზომები

ამ სექციაში ჩვენ განვმარტავთ სიტყვების მსგავსების ახალ საზომ ერთეულებს, რომლებიც დაფუძნებული იქნებიან ფრაზების n-გრამიან მსგავსებაზე. მაგრამ, სანამ განვმარტავდეთ, აუცილებელია გავითვალისწინოთ ორი მნიშვნელოვანი საკითხი: ნორმალიზაცია და სიმბოლოების გამყარება.

4.1. სიტყვების მსგავსების ნორმალიზაცია

ორი სიტყვის შედარებისას, საერთო ქვე-ფრაზის სიგრძე სიტყვების სიგრძის ზრდასთან ერთად იზრდება. იმისათვის, რომ თავიდან ავიცილოთ დამოკიდებულება მსგავსების კოეფიციენტისა და შესადარებელი სიტყვების ზომის, აუცილებელია შემოვიღოთ მსგავსების ნორმალიზირებული ვარიანტი.

LCS-ს გამოყენებისასაც, უმრავლეს შემთხვევაში უპირატესობას ნორმალიზირებულ ვარიანტს ანიჭებენ (LCSR). LCSR - უდიდესი საერთო ქვე-ფრაზის ფარდობა (Longest common Subsequence Ratio) გამოითვლება შემდეგნაერად: მას შემდეგ, რაც ვიპოვით უდიდეს საერთო ქვე-ფრაზის სიგრძეს (LCS სიგრძე) ვყოფთ მას შესადარებელ სტრინგებს შორის უდიდესის სიგრძეზე [8].

EDIT - რედაქტირების დისტანციაც ხშირად მსგავსი პრინციპით ნორმალიზირდება, ანუ უდიდესი სიტყვის სიგრძეზე გაყოფით (მაგალითად [5]). სწორედ ამ ხერხს ვიყენებთ ჩვენც n-გრამიანი მსგავსებისა და დისტანციის ნორმალიზირებისას.

მარაზალი და ვიდალი (Marazal, Vidal [6]) იმის მაგივრად, რომ ნორმალიზება უდიდესი სტრინგის სიგრძეზე გაყოფით მოხდეს, გვთავაზობენ ნორმალიზება

დასარედაქტირებელი ქვე-ფრაზის სიგრძით მოხდეს, რაც შედარებით უფრო რთულ ალგორითმს მოითხოვს.

4.2. სიმბოლოების გამყარება

სიმბოლოების გამყარება არის ხერხი, რომლის მეშვეობითაც ჩვენ ვზრდით სიტყვის საზღვრებზე მყოფი სიმბოლოების სენსიტიურობას. საზღვრებზე მდებარე სიმბოლოები იღებენ ნაკლებ n-გრამებში მონაწილეობას ვიდრე სიტყვის შიგნით მდგარი სიმბოლოები, რაც რა თქმა უნდა, არასწორია, მით უმეტეს, როდესაც ადამიანისთვის სიტყვის აღქმისას, ქართულ ენაში, ისევე, როგორც ნებისმიერ სხვა ენაში, დიდ ზეგავლენას ახდენს დასაწყისში მყოფი სიმბოლოები. მაგალითისთვის, სიტყვა *abc* შეიცავს ორ ცალ ბიგრამს: *ab* და *bc,* როგორც ვხედავთ, სიტყვის შუაში მყოფი სიმბოლო *b* ორ ბიგრამში ღებულობს მონაწილეობას, ხოლო მის საზღვრებზე მდებარე სიმბოლოები *a* და *c* კი მხოლოდ თითო ბიგრამში. როგორც უკვე ვთქვით, სიტყვების მსგავსების შედარების დროს აუცილებელია ყველა სიმბოლო თანაბრად უნდა მონაწილეობდეს შედარების პროცესში, და ამ პრობლემის თავიდან ასაცილებლად, ხანდახან, ხელოვნურად ხდება სიტყვების თავსა და ბოლოში სიმბოლოების დამატება, რათა მოხდეს საზღვარზე მყოფი *სიმბოლოების გამყარება.*

ჩვენ სიმბოლოების გასამყარებლად სიტყვების თავში ვამატებთ სიმბოლოებს, რამეთუ, ჩვენი აზრით, სიტყვის საწყისი სიმბოლოები ყველაზე მნიშვნელოვან როლს თამაშობს ადამიანისთვის სიტყვის აღქმისას ე.ი. სიტყვების მსგავსებისას. ყოველი სიმბოლოსთვის ანბანიდან ვარჩევთ უნიკალურ სხვა სიმბოლოს და ვამატებთ სიტყვის თავში n-გრამიანი მსგავსებისას *n-1* ცალ სიმბოლოს. უნიკალურ სიმბოლოებად ქართული ანბანის ყოველი ასოსთვის ვიღებთ შესაბამის ლათინურ ასოებს, მაგალითად თუ გვაქვს სიტყვა „სამშაბათი“, და გვინდა მისი შედარება n-გრამიანი მსგავსებით, როდესაც *n=3*, მაშინ სიმბოლოების გამყარების შემდეგ მივიღებთ სიტყვას „ssსამშაბათი“. იმის გათვალისწინებით, რომ თავდაპირველი დასადარებელი სიტყვების სიგრძეები იყო K და L შესაბამისად, n-გრამების რაოდენობა გაიზარდა $K + L - 2(n - 1)$ დან $K + L$ -მდე.

სიმბოლოების გამყარებისა და ნორმალიზაციის (მსგავსების კოეფიციენტის გაყოფა max(K,L) - უდიდესი სიტყვის სიგრძეზე) შემდეგ, ჩვენს მიერ შექმნილი ახალი მსგავსების შეფასება უეჭველად დააბრუნებს 1 მაშინ და მხოლოდ მაშინ, როდესაც სიტყვები ერთნაერია, და დააბრუნებს 0-ს მაშინ, როდესაც სიტყვებს არცერთი ასო არ აქვთ საერთო.

შემოთავაზებული n-გრამიანი მსგავსებისა და დისტანციის მეთოდების ალგორითმული გადაწყვეტა, რომლებშიც გათვალისწინებულია სიტყვებში საწყისი სიმბოლოების გამყარება და ასევე შედეგის ნორმალიზაცია უდიდესი ფრაზის სიგრძით წარმოდგენილია ნაშრომის ბოლოს დანართ I, დანართ II-სა და დანართ III-ის სახით.

დასკვნა

ჩვენ ჩამოვაყალიბეთ ახალი ცნება N-გრამიანი მსგავსებისა და დისტანციისა, რომელიც სტანდარტული უნიგრამული მსგავსების და დისტანციის განვრცობილი ვარიანტია. მის ბაზაზე ჩვენ ჩამოვაყალიბეთ სიტყვების მსგავსების ფორმალური საზომი ერთეულები. ჩვენ წარმოვადგინეთ N-გრამიანი მსგავსებისა და დისტანციის ფორმალური, რეკურსიული და ალტერნატიული განმარტება. ასევე ჩამოვაყალიბეთ მათი უფრო დახვეწილი ვარიაციები - ყოველმხრივი და პოზიციური მიდგომა მსგავსებისა და დისტანციის შესაბამისად. ასევე წარმოვადგინეთ N-გრამიანი მსგავსებისა და დისტანციის გამოთვლის ალგორითმები და პროგრამული უზრუნველყოფა. ჩვენ ასევე დავასკვენით, რომ საერთო ჯამში, შედეგის ნორმალიზაცია უდიდესი სიტყვის ზომით აუცილებლობას წარმოადგენს. ჩვენი აზრით, ტესტებმა უნდა აჩვენონ, რომ N-გრამიანი მსგავსება და დისტანცია უკეთ გაართმევს სიტყვების მსგავსების დადგენის ამოცანას თავს, ვიდრე კლასიკური სტანდარტული უნიგრამული მსგავსება და დისტანცია. საბოლოო ჯამში, შევქმენით სატესტო პროგრამა, რომელიც იღებს ორ სტრიქონს შენატანად და აბრუნებს მათი მსგავსებისა და დისტანციის N-გრამიან კოეფიციენტებს.

ჩვენი აზრით, შექმნილი პროდუქტი დამაკმაყოფილებელ დროში ახერხებს ამოცანის შესრულებას და ასევე შედეგად ადეკვატურ მსგავსებისა და დისტანციის კოეფიციენტს იძლევა.

იმის გათვალისწინებით, რომ ჩვენ მოვახდინეთ პირველადი n-გრამიანი მსგავსებისა და დისტანციის ალგორითმების რეალიზაცია, შემდგომი განვითარებისთვის შესაძლებელია მათი დახვეწილი ვარიანტების ყოველმხრივი და პოზიციური N-გრამიანი მსგავსებისა და დისტანციის, რომელთა ფორმულირებაც მოყვანილი გვაქვს ნაშრომში, ალგორითმისა და პროგრამული უზრუნველყოფის შექმნა.

N-გრამიანი მსგავსება და დისტანციის ალგორითმები და პროგრამული უზრუნველყოფა ჯერ კიდევ დატესტვისა და შემუშავების სტადიაშია. ჩვენ ვახდენთ სხვადასხვა სახის მონაცემების ტესტირებას და შედეგებს ვადარებთ სხვა ცნობილ ფრაზების შედარების ალგორითმებსა და შეფასების სისტემებს (როგორებიც არის DICE, XXDICE, LCSR).

გამოყენებული ლიტერატურა

1. Chris Brew and David McKelvie. 1996. Word-pair extraction for lexicography. In Proc. of the 2nd Intl Conf. on New Methods in Language Processing, pages 45–55.
2. Vaclav Chvatal and David Sankoff. 1975. Longest common subsequences of two random sequences. Journal of Applied Probability, 12:306–315.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. Introduction to Algorithms. The MIT Press, second edition.
4. John Hewson. 1993. A computer-generated dictionary of proto-Algonquian. Hull, Quebec: Canadian Museum of Civilization.
5. Bruce L. Lambert, Swu-Jane Lin, Ken-Yu Chang, and Sanjay K. Gandhi. 1999. Similarity as a Risk Factor in Drug-Name Confusion Errors: The Look-Alike (Orthographic) and Sound-Alike (Phonetic) Model. Medical Care, 37(12):1214–1225.
6. A. Marzal and E. Vidal. 1993. Computation of normalized edit distance and applications. IEEE Trans. Pattern Analysis and Machine Intelligence, 15(9):926–932.
7. I. DanMelamed. 1998. Manual annotation of translational equivalence: The Blinker project. Technical Report IRCS #98-07, University of Pennsylvania.
8. I. Dan Melamed. 1999. Bitext maps and alignment via pattern recognition. Computational Linguistics, 25(1):107–130.
9. D. Sankoff and J. B. Kruskal, editors. 1983. Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. Addison-Wesley.
10. Bill Smyth. 2003. Computing Patterns in Strings. Pearson.
11. Dan Tufis. 2002. A cheap and fast way to build useful translation lexicons. In Proc. of the 19th Intl Conf. on Computational Linguistics, pages 1030–1036.
12. Esko Ukkonen. 1992. Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science, 92:191–211.
13. Use caution — avoid confusion. United States Pharmacopeial Convention Quality Review, No. 76, March 2001.
14. Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. Journal of the Association for Computing Machinery, 21(1):168–173.
15. Grezegorz Kondrak. String Processing and Information Retrieval, Lecture Notes in Computer Science Volume 3772, 2005.

დანართი I:

საერთო ფუნქციები - ალგორითმი, პროგრამული შესრულება

```
//ფუნქცია MAX
//ფუნქცია აბრუნებს მაქსიმალურს სამ double ტიპის a, b და c პარამეტრიდან
// დასაბრუნებელი ტიპი double

private double MAX(double a, double b, double c)
{
      if (a >= b && a >= c)
            return a;
      else if (b >= a && b >= c)
            return b;
      else return c;
}
```

```
//ფუნქცია MAX
//ფუნქცია აბრუნებს მაქსიმალურს ორ int ტიპის a და b პარამეტრიდან
// დასაბრუნებელი ტიპი int

private int MAX(int a, int b)
{
      if (a >= b)
            return a;
      return b;
}
```

```
//ფუნქცია MIN
//ფუნქცია აბრუნებს მინიმალურს სამ double ტიპის a, b და c პარამეტრიდან
// დასაბრუნებელი ტიპი double

private double MIN(double a, double b, double c)
{
      if (a <= b && a <= c)
            return a;
      else if (b <= a && b <= c)
            return b;
```

```
        else return c;
}

/***************************************************/
/*          ფრაზის აფიქსაციის (სიმბოლოების გამყარების) ფუნქცია           */
/*          არგუმენტები:                                    */
/*          X - ფრაზა, რომლის აფიქსაციაც გვსურს ( string ტიპის )                 */
/*          დასაბრუნებელი ტიპი string - აფიქსირებული სიტყვა           */
/*    ფუნქცია ქართულ სიმბოლოებს ამყარებს შესაბამისი ინგლისური სიმბოლოებით     */
/*    ფუნქცია ინგლისურ სიმბოლოებს ამყარებს მთავრულ სიმბოლოებს - ინგლისური */
/*    შესაბამისი მხედრულით  ხოლო მხედრულ სიმბოლოებს შესაბამისი ინგლისური */
/*         მთავრულით         */
/***************************************************/

private string AffixeString(string X)
{
      if (!string.IsNullOrEmpty(X))
            switch (X[0])
            {
                  case 'ა': X = "a" + X;
                        break;
                  case 'ბ': X = "b" + X;
                        break;
                  case 'გ': X = "g" + X;
                        break;
                  case 'დ': X = "d" + X;
                        break;
                  case 'ე': X = "e" + X;
                        break;
                  case 'ვ': X = "v" + X;
                        break;
                    case 'ზ': X = "z" + X;
                        break;
                    case 'თ': X = "T" + X;
                        break;
                    case 'ი': X = "i" + X;
                        break;
                    case 'კ': X = "k" + X;
                        break;
                    case 'ლ': X = "l" + X;
                        break;
                    case 'მ': X = "m" + X;
                        break;
                    case 'ნ': X = "n" + X;
                        break;
                    case 'ო': X = "o" + X;
                        break;
```

```
                case 'პ': X = "p" + X;
                    break;
                case 'ჟ': X = "J" + X;
                    break;
                case 'რ': X = "r" + X;
                    break;
                case 'ს': X = "s" + X;
                    break;
                case 'ტ': X = "t" + X;
                    break;
                case 'უ': X = "u" + X;
                    break;
                case 'ფ': X = "f" + X;
                    break;
                case 'ქ': X = "q" + X;
                    break;
                case 'ღ': X = "R" + X;
                    break;
                case 'ყ': X = "y" + X;
                    break;
                case 'შ': X = "S" + X;
                    break;
                case 'ჩ': X = "C" + X;
                    break;
                case 'ც': X = "c" + X;
                    break;
                case 'ძ': X = "Z" + X;
                    break;
                case 'წ': X = "w" + X;
                    break;
                case 'ჭ': X = "W" + X;
                    break;
                case 'ხ': X = "x" + X;
                    break;
                case 'ჯ': X = "j" + X;
                    break;
                case 'ჰ': X = "h" + X;
                    break;
                default: {
                     if(char.IsUpper(X[0]))
                           X =  char.ToLower(X[0]).ToString() + X;
                     else
                           X = char.ToUpper(X[0]).ToString() + X;
                     Break;
                  }
        }
    return X;
```

```
}
```

დანართი II:

N-გრამიანი მსგავსება ალგორითმი და პროგრამული უზრუნველყოფა

```
/**************************************************/
/*        N-გრამიანი მსგავსების დამთვლელი ფუნქცია        */
/*        არგუმენტები:                                   */
/*        X - პირველი სიტყვა ( string ტიპის )            */
/*        Y - მეორე  სიტყვა ( string ტიპის )             */
/*        N  -  გრამის  სიგრძე ( int ტიპის )             */
/*        დასაბრუნებელი ტიპი double - მსგავსების კოეფიცინეტი        */
/**************************************************/

private double NGramSim(string X, string Y, int N)
{
      //სიტყვების სიგრძეების ინიციალიზაცია

      int K = X.Length;
      int L = Y.Length;


      //მასივის გამოცხადება

      var S = new double[K + 1][];
      for (var i = 0; i <= K; i++)
      {
            S[i] = new double[L + 1];
      }


      //სიტყვების საწყისი სიმბოლოების გამყარება

      for (int u = 1; u < N; u++)
      {
            X = AffixeString(X);
            Y = AffixeString(Y);
      }


      //მასივიში ნულების ინიციალიზაცია

      for (var i = 0; i <= K; i++)
      {
```

```
            S[i][0] = 0;
        }


        //მასივში ნულების ინიციალიზაცია

        for (var i = 0; i <= L; i++)
        {
            S[0][i] = 0;
        }



        //ალგორითმი დინამიური პროგრამირებით

        for (int i = 1; i <= K; i++)
        {
          for (int j = 1; j <= L; j++)
          {
              int SN = 0;
              if (!(X.Length - i < N - 1 || Y.Length - j < N - 1))
              {
                  SN = X.Substring(i - 1, N) == Y.Substring(j - 1, N) ? 1 : 0;
              }

              S[i][j] = MAX(S[i - 1][j], S[i][j - 1], S[i - 1][j - 1] + SN);
           }
        }

        return S[K][L] / MAX(K, L);
}
```

დანართი III:

N-გრამიანი დისტანცია - ალგორითმი, პროგრამული შესრულება

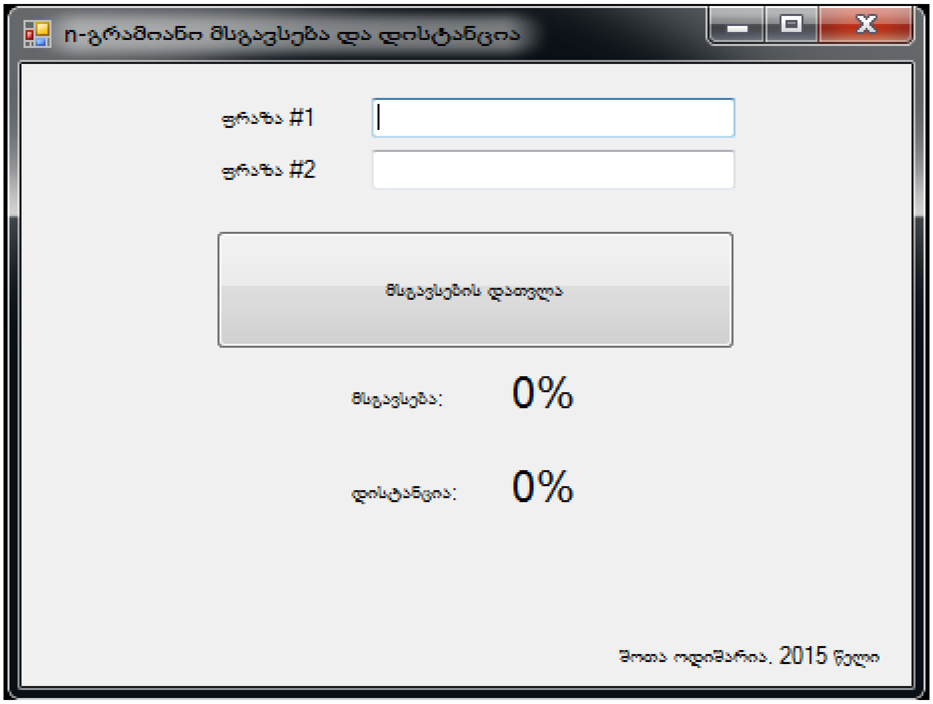
```
/*************************************************/
/*        N-გრამიანი დისტანციის დამთვლელი ფუნქცია      */
/*        არგუმენტები:                                  */
/*        X - პირველი სიტყვა ( string ტიპის )           */
/*        Y - მეორე  სიტყვა ( string ტიპის )            */
/*        N  -  გრამის  სიგრძე ( int ტიპის )            */
/*        დასაბრუნებელი ტიპი double - დისტანცია         */
/*************************************************/

private double NGramDist(string X, string Y, int N)
{

    //სიტყვების სიგრძეების შევსება

     int K = X.Length;
     int L = Y.Length;


     //მასივის გამოცხადება

     var D = new double[K + 1][];
     for (var i = 0; i <= K; i++)
    {
          D[i] = new double[L + 1];
    }


     //სიტყვების საწყისი სიმბოლოების გამყარება

     for (int u = 1; u < N; u++)
     {
          X = AffixeString(X);
            Y = AffixeString(Y);
    }


     //მასივში ნულების ინიციალიზაცია

     for (var i = 0; i <= K; i++)
     {
          D[i][0] = i;
    }
```

```
        //მასივიში ნულების ინიციალიზაცია

        for (var i = 0; i <= L; i++)
        {
             D[0][i] = i;
       }


       //ალგორითმი დინამიური პროგრამირებით

        for (int i = 1; i <= K; i++)
        {
          for (int j = 1; j <= L; j++)
          {
             int DN = 1;
             if (!(X.Length - i < N - 1 || Y.Length - j < N - 1))
               DN = X.Substring(i - 1, N) == Y.Substring(j - 1, N) ? 0 : 1;

             D[i][j] = MIN(D[i - 1][j] + 1, D[i][j - 1] + 1, D[i - 1][j - 1] +
             DN);
           }
       }

       return D[K][L] / MAX(K, L);
}
```

დანართი VI:

N-გრამიანი დისტანციისა და მსგავსების მთვლელი პროგრამის ტესტები

პროგრამის ინტერფეისი:



სატესტო პროგრამა დაწერილია C# ენაზე, Windows Forms Application GUI-ის გამოყენებით, .Net Framework 4.5 სტრუქტურაში.

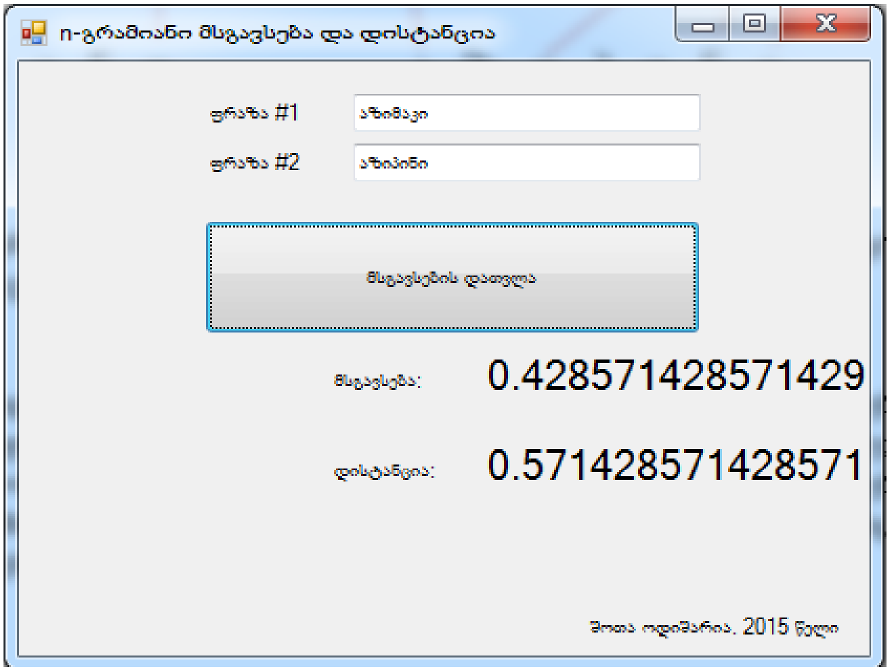
პროგრამა შენატანად იღებს ორ სიტყვას : „ფრაზა #1“ და „ფრაზა #2“-ს. მსგავსების დათვლის შემდეგ კი გამოაქვს სიტყვებს შორის მსგავსებისა და დისტანციის კოეფიციენტები, კოეფიციენტებში გათვალისწინებულია ზემოთ ხსენებული ნორმალიზაციისა და აფიქსაციის მეთოდები (საწყისი სიმბოლოების გამყარება).

პროგრამას გამოაქვს მაქსიმალური მსგავსება 1 და მინიმალური დისტანცია 0 - მაშინ და მხოლოდ მაშინ, როდესაც სიტყვები ტოლია.

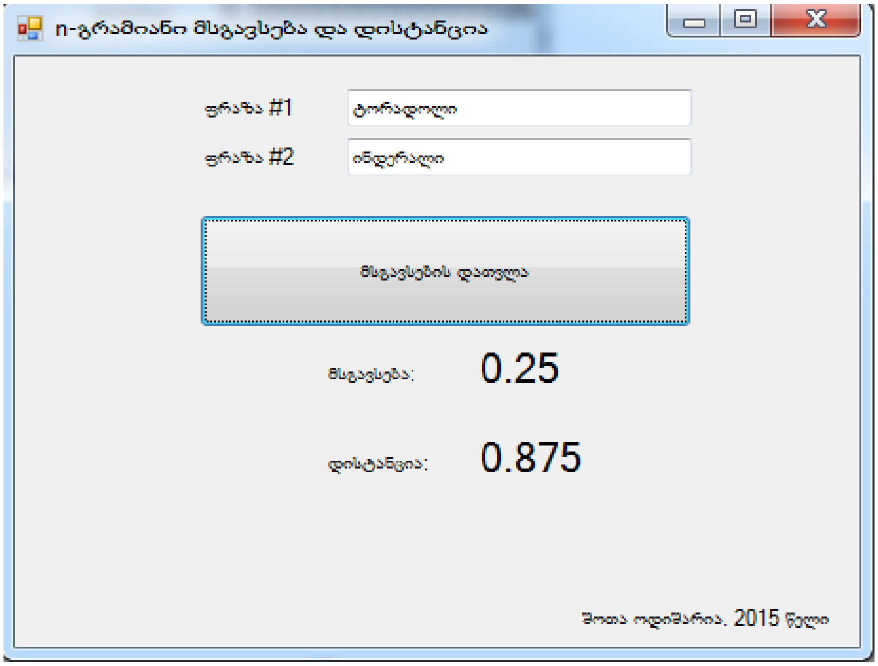
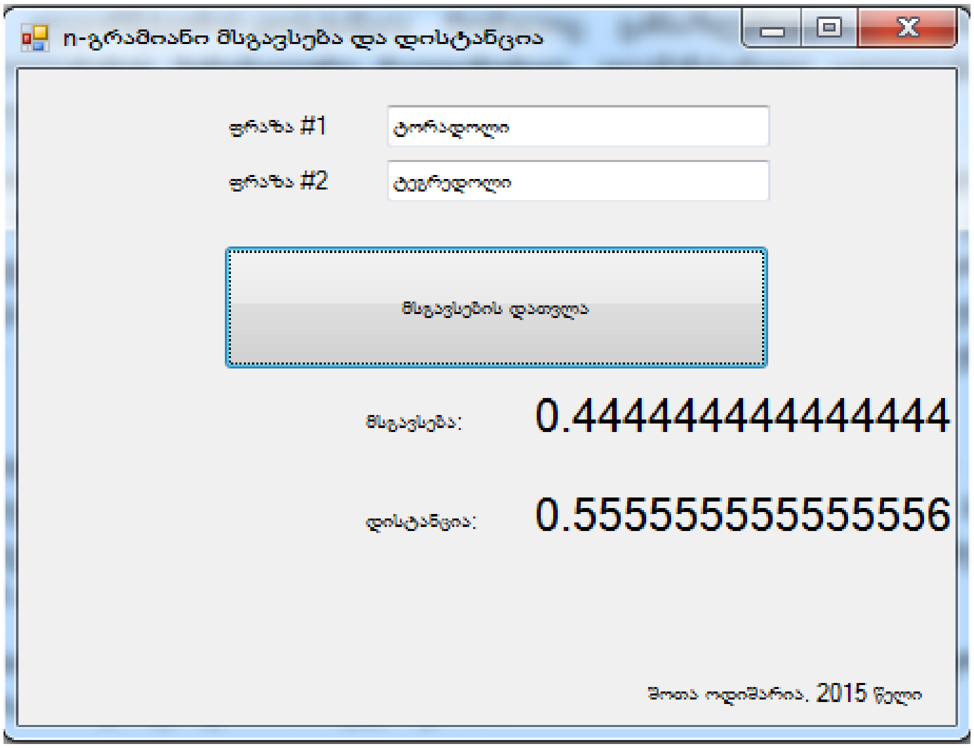
ასევე პროგრამას გამოაქვს მინიმალური მსგავსება 0 და მაქსიმალური დისტანცია 1 მაშინ და მხოლოდ მაშინ, როდესაც სიტყვებში არცერთი n-გრამი არ არის ტოლი.

პროგრამის მუშაობის მაგალითები:

აზიმაკი - აზიპინი



ტორადოლი - ინდერალი



ტორადოლი - ტეგრედოლი