Covid: een kreet om hulp

Freya Van Speybroeck, student BoSc informatica Ugent

**ABSTRACT**

*Dit is een korte samenvatting over het artikele dat op 1 kolom moet staan en dan de inleiding is gewoon blablabla TODO: iets met aarde is plat vs rond als voorbeeld of metafoor*

**INLEIDING**

SARS-CoV-2 heeft het afgelopen jaar de wereld op stelten gezet. Maar is het een onschuldige mutatie, of is er iets anders aan de hand? Er zijn al veel studies gedaan omtremt het virusch, waarbij naar het genetisch materiaal gekeken wordt. Zo kan er gekeken worden naar het proteoom van SARS-CoV-2, dat bestaat uit 17 eiwitten [1]. Om dit in perspectief te brengen: het menselijk proteoom telt maar liefst 75 776 eiwitten [2].

Het proteoom van het virus is de focus van dit artikel. Elk eiwit in het proteoom bestaat uit lange kettingen van aminozuren, elk voorgesteld door een hoofdletter. We werken dus met een reeks van letters, waar er eventueel woorden uit te halen zijn. Een succesvolle techniek om verborgen boodschappen uit tekst te vinden, werd al toegepast op de bijbel [3]. Het zoeken naar woorden beginnende uit een willekeurige startpositie, en dan vooruit of achteruit sprongen maken van vaste afstand, heeft al geleid tot de voorspelling van een aantal belangrijke gebeurtenissen in de geschiedenis.

Dit toepassen op genetisch materiaal is dus ook perfect mogelijk, en precies wat we verder zullen uitwerken.

**Materialen en methoden**

Het doel is om uit een woordenboek van ongeveer 350.000 woorden, verborgen boodschappen te vinden in het SARS-CoV-2 proteoom [1]. Merk wel op dat de letters B, J, O, U, X en Z niet overeenkomen met een aminozuur. Volgens een betrouwbare bron [4] kunnen deze letters als volgt geinterpreteerd worden:

* B stelt N of D voor.
* Z stelt Q of E voor.
* X en O kunnen elke letter zijn.
* J en U staan voor resp. I en V.

Om een woord in een eiwit te vinden, moeten alle startposities, van daaruit alle sprongroottes (vooruit en achteruit) en daarvan alle deelstrings overlopen worden.

Een makkelijke oplossing zou zijn om het woordenboek in een set [5] op te slaan, die aan de hand van hashing in constante tijd zou moeten kunnen kijken of een element erin zit. Dan is het simpel: overloop alle deelstrings van het eiwit (zoals hierboven besproken) en kijk of die in de set zitten, zo ja, voeg ze toe aan de lijst van gevonden woorden.

Het word ingewikkelder als er rekening moet worden gehouden me de speciale letters B, Z, X, O, J en U. Om dit op te lossen, zouden we voor woorden met deze letters, alle alternatieven kunnen genereren.

De gebruikte methode werkt niet op deze manier, maar werkt met tries [6]. De trie is een datastructuur die zeer geschikt is voor het werken met strings. Om te zorgen dat de speciale letters juist worden afgehandeld, kunnen we dus een gewone trie nemen, en die aanpassen voor onze doeleinden.

Om zo goed mogelijk gebruik te maken van de trie, kunnen we ook het verwerken van de deelstrings van het eiwit anders aanpakken, namelijk als volgt. Neem alle langste strings van verschillende startposities en verschillende sprongroottes (enkel vooruit). Eerst steken we die strings in een set, zodat we zo weinig mogelijk duplicaten moeten controleren. Van die strings willen we nu alle deelstrings bekijken (in 2 richtingen). In plaats van elke mogelijke deelstring appart te zoeken, kunnen we ze letter per letter verwerken, en stoppen vanaf we merken dat er in de trie niet meer verder gezocht kan worden.

Om dit te kunnen doen, hebben we een zoekmethode nodig zodat in de trie letter per letter gezocht kan worden door een zoekfront bij te houden. Op die manier kunnen we steeds de substring uitbreiden met de volgende letter, zonder helemaal opnieuw in de trie te moeten beginnen zoeken. Die methode ziet er als volgt uit:

functie find\_update(letter):

resultaat = []

speciale\_letter = B, Z, J of U // enkel als letter = N,D,Q,E,I of V

te\_zoeken\_letters = [letter, ‘X’, ‘O’, speciale\_letter ]

nieuw\_zoekfront = []

Zolang trie.zoekfront niet leeg:

n = zoekfront.pop()

nodes = de kinderen van n die 1 van de te\_zoeken\_letters voorstellen

nieuw\_zoekfront += nodes

Resultaat += [ node.woord for node in nodes als node een woord bevat ]

Als nieuw\_zoekfront leeg is:

Return ([], False)

Trie.zoekfront = nieuw\_zoekfront

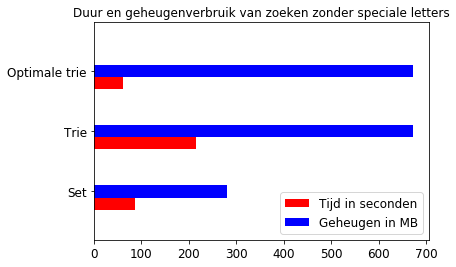
Return(resultaat, True)

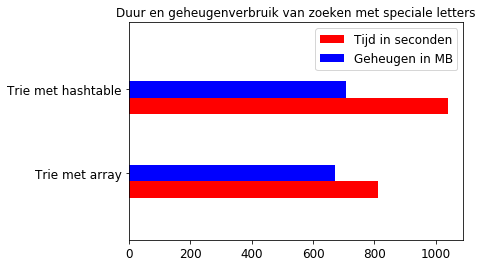
We stoppen pas vanaf er geen nodes meer in het zoekfront zijn, niet als een bepaalde string geen woord is. Je kan dus een pad aflopen waarbij nodes soms een woord bevatten, en soms niet. Vanaf er gestopt wordt, word een andere startpositie van de grote string genomen, en gaan we zo verder.

Om te zorgen dat we ook achterwaartse sprongen controleren, bouwen we ook de deelstring op startende achteraan de string, en zo naar voor gaande. Daarbij zorgen we dat enkel substrings van lengte minimaal 2 overlopen worden, de letters doen we op het einde nog eens appart.

**Resultaten en discussie**

Om ons van de uitgelegde aanpak te overtuigen, kunnen we verschillende tijdsmetingen doen. Hiervoor zullen we vergelijken zonder de speciale letters, om een zicht te hebben van hoe de datastructuren zich gedragen in normale situaties. In de figuur zien we “geoptimaliseerde trie” als de aanpak die hierboven is uitgelegd, “set” werkt zoals uitgelegd in de simpele aanpak, en “Trie” werkt met een gewone opzoek methode die het hele woord nodig heeft om te zoeken, en dus niet letter per letter een zoekfront gaat updaten.

We zien dat werken met een gewone trie, veel nadeliger is dan werken met een simpele set. Maar het vroegtijdig kunnen stoppen voor een deelstring, werkt wel nog sneller dan een set. Er is wel meer geheugen nodig voor werken met tries.



De resultaten toonden iets opvallends aan. In de 17 eiwitten kwam 16 keer het woord “SOS” voor. Verder in de woorden vonden we 12 keer “LAT” en 8 keer “LONG”. Deze lijken sterk op de afkortingen van “latitude” en “longitude”. In volgende tabel zien we de nummers van eiwitten waarin “LAT” en “LONG” voorkomen (tellen start vanaf 0).

|  |  |
| --- | --- |
| LAT | 1,2,3,5,6,7,10,11,12,14,15,16 |
| LONG | 2,3,6,10,11,12,14,15 |

Als we nu dezelfde methode gebruiken om geheime boodschappen te vinden, zien we dat bij de “LAT” een sprong grootte van 4 de getallen 3, 10 en 15 oplevert. Als we de 1’en weghalen en de volgorde behouden, krijgen we 305. Hetzelfde kunnen we doen voor “LONG”, als we bij 11, 12 en 14 een paar cijfers weglaten is het resultaat 114. De cijfers 305 en 114 komen je misschien niet bekend voor, maar dit zijn zeer niet-toevallig de eerste 3 cijfers van de coordinaten van Wuhan [7]: 30.57° N, 114.27 ° E.

Dit begint heel sterk te lijken op een noodsignaal, gecodeerd in het genetisch materiaal van het virus, dat zijn start kende in Wuhan. Is het virus gefabriceerd door iemand die in gevangenschap wordt gehouden? Door een onderzoeker die ons wilt waarschuwen? Natuurlijk kunnen we dus de overheid van China niet vertrouwen, maar evenmin de andere landen, die nu een voor een vaccins aan het uitbrengen zijn. Hoe kon zo een simpele boodschap als dit gemist worden? Er is duidelijk iets groters aan de hand, laat dit artikel dus een waarshuwing zijn voor wat te komen is.

**Literatuur**

[1]https://www.uniprot.org/proteomes/UP000464024

1. <https://www.uniprot.org/proteomes/UP000005640>
2. <https://en.wikipedia.org/wiki/Bible_code>
3. <https://ufora.ugent.be/d2l/le/content/237612/Home>
4. <https://docs.python.org/3/tutorial/datastructures.html#sets>
5. <https://en.wikipedia.org/wiki/Trie>
6. https://latitude.to/articles-by-country/cn/china/1395/wuhan

python -m cProfile -o profile antivaxxer.py bwoordenboek.txt covid.fa

We hebben een grote string “HANDEN” waarvan we willen kijken of alle deelstrings erin zitten. De trie bevat enkel de woorden “HAN” en “HON”.

We beginnen op startindex 0, en lezen dus “H” in, updaten het zoekfront in de trie, en geven dan terug of “H” werd gevonden. Het zoekfront bevat nu alle nodes die matchen met “H”, wat momenteel maar 1 zal zijn (het is geen speciale letter). We lezen nu “A” in, en updaten het zoekfront van de trie. “H” wordt dus verwijderd uit het zoekfront en we zoeken nu de “A” in zijn kinderen. We voegen het kind “A” toe, maar voegen ook het kind “O” toe, aangezien die speciale letter met alles overeenkomt. Er zitten dus nu 2 nodes in het zoekfront van de trie. De volgende letter is “N”, die in beide nodes van het zoekfront werden gevonden. Deze 2 “N” nodes zullen ook aangeven dat ze een woord bevatten, namelijk “HON” bij de ene en “HAN” bij de andere, die we dus aan de matches kunnen toevoegen. De letter “D” zullen we jammer genoeg niet vinden, waardoor we kunnen stoppen met zoeken, en met zekerheid kunnen zeggen dat noch “HAND”, “HANDE”, of “HANDEN” een match zal opleveren.