Universitat de Lleida

Escola Politècnica Superior

El repte del SARS-CoV-2

Pràctica Algorísmia i Complexitat

Noms:
Víctor Alcobé
Tibireu Paiu
Dand Marbà
Pau Agustí

Data:
25 de Maig de 2020

Índice

1. Introducció

Aquesta pràctica que hem implementat en l'assignatura d'Algorísmia i Complexitat s'ha basat en la situació actual que estem vivint, la pandèmia del COVID-19. Un dels grans problemes dels coronavirus és la mutació del seu ARN.

L'objectiu és aconseguir estudiar les diferents mostres per a poder classificar-les i així identificar que parts de l'ARN han mutat en cada classe. Així doncs, nosaltres pas a pas, després de tractar la informació extreta de la base de dades de *National Center of Biotechnology Information*, realitzar el càlcul de la mediana de longitud, per poder tenir una única mostra per país, i de realitzar un algoritme per alineament de seqüències, hem pogut arribar a classificar les diferents mostres del SARS-CoV-2 utilitzant *K-Means Clustering*.

1.1. Distribució de tasques

Per la realització de la pràctica vam començar utilitzant l'eina especificada en l'enunciat, *GitHub*, i distribuint-nos feina, però després d'uns dies treballant així vam ser capaços de coordinar-nos per a poder quedar cada setmana diferents dies per poder realitzar la pràctica en comú. Vam decidir treballar així perquè tothom fos capaç d'entendre la totalitat de la pràctica, i no que una parella realitzés la classificació sense entendre l'alineament de seqüències i a l'inrevés.

L'eina utilitzada per poder realitzar les reunions i treballar en grup va ser *Discord*, el qual ens permetria xat de veu i vídeo, i la retransmissió de la nostra pròpia pantalla, juntament amb la compartició d'arxius.

2. Preprocessament

En la primera part de la pràctica, hem de calcular la mediana de longitud de totes les mostres per país. Seguidament, quan tinguem les mostres de longitud mediana per cada país, descarregarem els fitxers *FASTA* de cada una de les mostres de longitud mediana, fitxers que necessitarem per la següent part de la pràctica.

2.1. Anàlisi de dades a tractar

Les dades a tractar en aquesta pràctica es divideixen en dues classificacions, per un costat tenim els fitxers *FASTA* dels quals parlarem més endavant, i per altra banda tenim un fitxer *CSV*. Aquest fitxer és descarregat del *National Center of Biotechnology Information*, i conté molta informació de totes les mostres que hi ha sobre el virus, indicant procedència, longitud, i moltes altres variables. Nosaltres sol ens fixarem en la procedència "*GeoLocation*", en la longitud de la mostra "*Length*" i en l'identificador de la mostra "*Accession*".

2.2. Disseny de la implementació

Els principals punts de la part de preprocessament d'aquesta pràctica, són:

2.2.1. Anàlisis de Dades

El fitxer analitzador de les dades, *parser.py*, en breus paraules, és l'encarregat de la lectura dels fitxers *CSV* i *FASTA*. El mètode principal de la classe *Parser* que conté aquest fitxer, *parse(),* s'encarrega de la lectura del fitxer *CSV*, fent ús de la llibreria *csv*. Aquest fitxer, també conté una funció que s'utilitzarà per a la lectura dels fitxers *FASTA* i realizarà l'extracció de les seqüències que seran necessàries en la segona part d'aquesta pràctica.

2.2.2. Algortime d'ordenació *QuickSort*

L'algoritme d'ordenació *QuickSort* del fitxer *median.py*, implementat en les funcions *sort()* i *partition(),* pseudocodi d'ambdues pot trobar-se posteriorment en el document, és utilitzat per l'ordenació de les longituds "*Lengths*" per a poder seleccionar-ne una de longitud mediana, en la funció que s'explicarà en la secció **Càlcul de la mediana**. Es tractarà més amb profunditat l'algoritme en la secció **Anàlisi de l'algoritme utilitzat**.

2.2.3. Càlcul de la mediana

Per a la realització del càlcul de mediana, s'ha implementat *calculate_median()* en el fitxer *median.py* per a poder seleccionar una mostra mediana d'una llista ordenada, en el nostre cas una mostra de longitud mediana. Es van separar els algoritmes d'ordenació i la funció de càlcul de mediana per a facilitar l'estructura del codi i millorar el cost.

2.2.4. Descàrrega dels fitxers *FASTA*

La descàrrega dels fitxers *FASTA* es realitza mitjançant la funció que es troba en el fitxer *sarscovhierarchy.py*, *download_accessions().* Aquesta funció rep com a paràmetre d'entrada una mostra de longitud mediana de cada país que hi ha en el fitxer *sequences.csv,* i descarrega els fitxers necessaris de la base de dades del *National Center of Biotechnology Information.*

2.3. Anàlisi de l'algoritme utilitzat

En aquesta secció es farà èmfasi en l'algoritme d'ordenació *QuickSort* utilitzat, com s'ha esmentat anteriorment.

2.3.1. Pseudocodi

Pseudocodi de l'algortime d'ordenació (*QuickSort*), i de la funció complementària (*Partition*) que es troben en el fitxer *median.py*.

Algorithm 1 Algoritme de Ordenació

```
1: procedure QuickSort(list, low, high, key)
2:     if low < high then
3:         pi ← Partition(list, low, high, key)
4:         QuickSort(list, low, pi − 1, key)
5:         QuickSort(list, pi + 1, high, key)
6:     end if;
7: end function;
```

Algorithm 2 Algoritme de Partició

```
1: procedure Partition(list, low, high, key)
2:     i ← low - 1
3:     pivot ← list[high][key]
4:     for j in range (low, high) do
5:         if list[j][key] <= pivot then
6:             i ← i + 1
7:             temp ← list[i]
8:             list[i] ← list[j]
9:             list[j] ← temp
10:        end if;
11:    end for;
12:    temp ← list[i + 1]
13:    list[i + 1] ← list[high]
14:    list[high] ← temp
15:    return i + 1
16: end function;
```

2.4. Estudi teòric

En l'estudi teòric primer comentarem el cost de l'algoritme QuickSort. Aquest té un cost mitjà de $O(n.log(n))$, tot i que en els pitjors dels casos, el cost pot arribar a ser de $O(n^2)$. Considerarem $O(n.log(n))$ el seu cost, on n és la longitud de la llista a ordenar.

També cal considerar altres costos en el preprocessament com:

- El cost de l'agrupació de la longitud per país que té un cost de $O(log(n))$ on n és el nombre de mostres totals que proporciona el fitxer CSV.
- El cost de la funció que calcula la mediana d'una llista que té un cost de $O(n)$ on n és la longitud de la llista.
- El cost de la lectura del fitxer que té un cost de $O(n)$ on n és la longitud del fitxer.

3. Alineament de seqüències

Per l'alineament de seqüències hem utilitzat l'algoritme Needleman-Wunsch, algortime que s'explicarà en la secció següent més detalladament. En breus paraules aquest algoritme mira el semblants que poden arribar a ser duess seqüències tenint en compte que pot haver-hi salts (gaps) per poder alinear les seqüències d'una forma òptima. La comparació de seqüències es realitza de dos en dos. El resultat cada duess seqüències s'emmagatzema per poder ser utilitzat en l'últim apartat de la pràctica.

3.1. Anàlisi d'algortimes existents

Per a poder trobar un alineament global en un temps raonable, s'han implementat nombrosos algoritmes. Com hem especificat anteriorment l'algoritme que s'ha utilitzat en aquesta pràctica per l'alineament global de seqüències ha set el Needleman-Wunsch, basat en programació dinàmica.

3.2. Disseny de la implementació

Els fitxers FASTA descarregats anteriorment, es llegeixen amb la funció parser_fasta() que es troba en parser.py. Després de llegir les seqüències, s'alineen aquestes de dos en dos utilitzant l'algoritme esmentat.

Essencialment l'algoritme utilitzat consisteix en:

1. Es defineix una funció de similitud entre els elements a realitzar l'alineament.
2. Els "indels" es penalitzen amb un cert pes.
3. Es construeix una matriu de mida (i+1 · j+1) on i i j són les longituds de les seqüències a alinear.

4. S'inicialitza la primera columna i fila amb 0. Aquest pot variar en diferents implementacions de l'algoritme.

5. És plena la matriu on els valors d'una cel·la depenen dels immediats anteriors, com s'indica en la fórmula següent.

$$matrix[i][j] \leftarrow Min \begin{matrix} matrix[i-1][j-1] + isEqual[seq1, seq2] \\ matrix[i-1][j] + gap \\ matrix[i][j-1] + gap \end{matrix}$$

6. El valor que es troba en la posició (x, y) on x i y sún la mida de la matriu és el valor de similitud.

El valor de similitud s'acaba calculant amb longitut_matriu - valor perquè el valor més petit sigui el valor de més similitud, i el valor més gran el de menys. Per tant si el valor és 0, significa que les seqüències són idèntiques.

En l'exemple de la Figura 3.1 es pot observar com quedaria la matriu amb les dos seqüències mostrades i amb els valors match = 1 (representat en la Figura 3.1. amb +), mismatch = 0 , gap = 0 (representat en la Figura 3.1. amb _), valors que poden ser modificables per diferentes resultats.

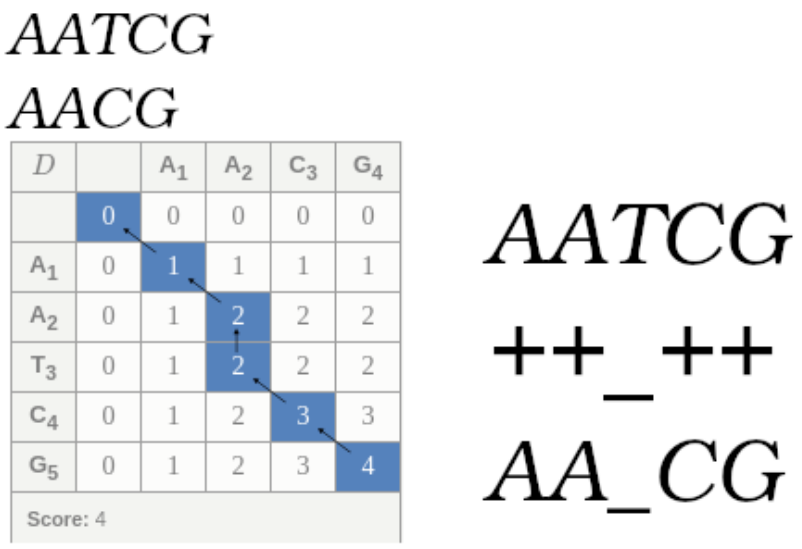

Figura 3.1.

3.2.1. Pseudocodi

Pseudocodi de l'algoritme Needleman-Wunsch utilitzat en la implementació de alignment.py.

Algorithm 3 Algoritme de Alineament

```
1: procedure MatrixGenerate(seq1, seq2, gap, mismatch, match)
2:     matrix[0][0] ← 0
3:     i ← 1
4:     for i to len(seq1) do
5:         matrix[i][0] ← seq1[i]
6:     end for;
7:     j ← 1
8:     for j to len(seq2) do
9:         matrix[0][j] ← seq2[j]
10:    end for;
11:    j ← 1
12:    j ← 1
13:    isEqual(True) ← match
14:    isEqual(False) ← mismatch
15:    for i to len(seq1)+1 do
16:        for j to len(seq2)+1 do
                                   matrix[i − 1][j − 1] + isEqual[seq1, seq2]
17:            matrix[i][j] ← Min  matrix[i − 1][j] + gap
                                   matrix[i][j − 1] + gap
18:        end for;
19:    end for;
20: end function;
```

3.3. Estudi teòric

El cost de l'algortime Needleman-Wunsch, és a dir, el cost de la creació de la matriu és $O(n.m)$ on n i m són respectivament les longituds de cada una de les seqüències a alinear.

El cost de l'alineament de la totalitat de les seqüències és de $O(n^2)$ on n és el nombre de seqüències. Cal destacar que per a reduir aquest cost a pràcticament la meitat, s'emmagatzemen les cadenes provades per evitar que si hem alineat Spain - Portugal, no alineéssim també Portugal - Spain.

4. Classificació

Aquesta última part de la pràctica, consta en classificar els resultats extrets en la part d'alineament.

4.1. Anàlisi d'algoritmes existents

Per poder classificar valors, existeixen una infinitat d'algoritmes. Entre totes les possibilitats vam elegir el K-Means Clustering, algoritme d'aprenentatge no supervisat, per la facilitat d'implementació d'aquest i per que un membre del grup ja va estudiar-lo de forma teòrica en una assignatura d'un curs superior.

4.2. Disseny de la implementació

L'algoritme K-Means Clusters, ens permetrà classificar les mostres analitzades anteriorment, en els grups que desitgéssim.A partir dels resultats de la part d'alineament, l'algoritme generarà els clusters amb valors aleatoris, aquests ens permeten fer una classificació grupal aproximada. Un cop realitzada la classificació observem que els clusters no estan perfectament alineats amb els grups el qual ens obliga a fer la mitjana dels valors que formen el grup del cluster, per tal de realitzar una classificació més precisa que l'anterior. Aquest procediment es realitza tants cops com iteracions indiquéssim.

En la Figura 4.1. pot veure's com els clusters estan posicionats en una iteració.

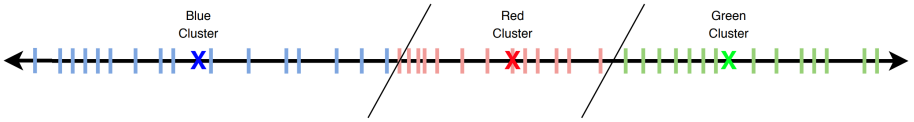

Figura 4.1.

En la Figura 4.2. pot observar-se com els clusters han set recalculats i lleugerament desplaçats, cosa que provoca que algunes de les mostres és reposicionin en els grups.

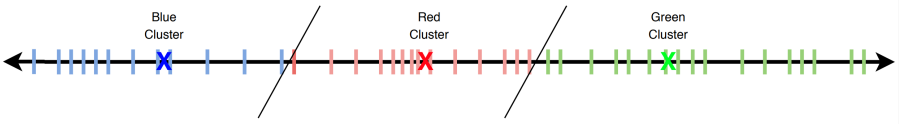

Figura 4.2.

4.2.1. Pseudocodi

Pseudocodi de l'lagoritme de K-Means Clustering de la implementació del fitxer kmeans_clustering.py

Algorithm 4 Algoritme de Classificació

```
1: procedure KMeansClustering(data, clusters, iterations)
2:     clusters ← generateRandomClusters()
3:     for i to iterations do
4:         for sample in data do
5:             cluster ← closestCluster(sample)
6:         end for;
7:         clusters ← computeClusters()
8:     end for;
9: end function;
```

4.3. Estudi teòric

L'algoritme de classificació té un cost de $O(m.n)$ on n és el nombre de mostres a classificar i m és el nombre d'iteracions introduïdes.

S'han de remarcar altres costos:

- La generació dels centroides de forma random que té un cost de $O(n)$ on n és el nombre de clusters a generar.
- La comprovació del cluster més proper té un cost de $O(n)$ on n és el nombre de clusters existents.
- La recalculació dels clusters té un cost de $O(n.m)$ on n és el nombre de clusters i m és el nombre de mostres que conté cada cluster.

5. Conclusions

La pràctica ha estat implementada seguint una metodologia basada en Programació Orientada a Objecte, Programació Dinàmica i BackTracking, cosa que a tots els membres del grup ens ha permès millorar i aprendre molt sobre programació, sobre algoritmes i com estudiar aquests, i sobre el llenguatge Python.

També volem remarcar que la pràctica ens ha semblat molt interessant i ha estat un punt fort que ha fet que treballar en ella no fos gens pesat, ja que proposava un projecte amb uns objectius que considerem que hem arribat a assolir.

Per últim volem comentar algun punt feble de la pràctica, com pot ser l'ús únicament de Python, ja que som conscients que amb algun llenguatge com Haskell

hauríem pogut reduir temps d'execució, i la impressió del resultat que per falta de temps no hem pogut usar cap llibreria perquè fos el resultat imprès d'una forma més gràfica.

6. Algunes especificacions sobre el codi

El que es dirà en aquest apartat pot trobar-se en el fitxer README.txt, però ho ficarem aquí també per deixar-ho més clar.

- La longitud de les seqüències analitzades és de 100, per evitar que per la correcció sigui necessària una espera llarga. Pot provar-se també amb seqüències majors variant la línia 58 del fitxer parser.py. El temps d'execució del programa amb longitud 1000, la recomanada a classe, es d'aproximadament 20 a 30 minuts depenen del sistema.
- Els clusters que es calculen en la classificació són 3 i es realitzen 100 iteracions per trobar els més precisos, aquests valors poden modificar-se en la línia 53 de l'arxiu sarscovhierarchy.py
- És necessària la instal·lació de numpy mitjançant pip install numpy o pip3 install numpy
- És recomanable usar ./sarscovhierarchy.py sequences.csv > result.txt per l'execució, per si el resultat és massa llarg per veure en consola, poder-lo emmagatzemar en un fitxer.
- És necessari donar permisos amb chmod u+x sarscovhierarchy.py.