

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL CIENCIA DE LA COMPUTACIÓN



BIOINFORMÁTICA

Examen N°2

GRUPO N°2

ALUMNOS:

APAZA CHAVEZ, MARIA LOURDES
BARRIOS CORNEJO, SELENE
GOMEZ CONTRERAS, JUNIOR VALENTIN
PILCO PANCCA, LUZ MARINA

DOCENTE:

MACHACA ARCEDA, VICENTE

AREQUIPA - PERÚ

2021

DESCRIPCIÓN:

En el presente trabajo se explica la implementación de nuestra aplicación Web, donde se pueda realizar el alineamiento de secuencias y generación de árboles filogenéticos. La aplicación sigue el siguiente proceso:

1. Alinear las secuencias
2. Generar las distancias
3. Muestra el árbol filogenético

Debido a que no se puede mostrar lo de la consola en web se uniformiza las entradas de los algoritmos para que puedan recibir entradas de .txt o imagen en su defecto. La aplicación hace uso de los siguientes algoritmos:

1. Alineamiento global (Needleman–Wunsch)

En consola se ejecuta de dos maneras:

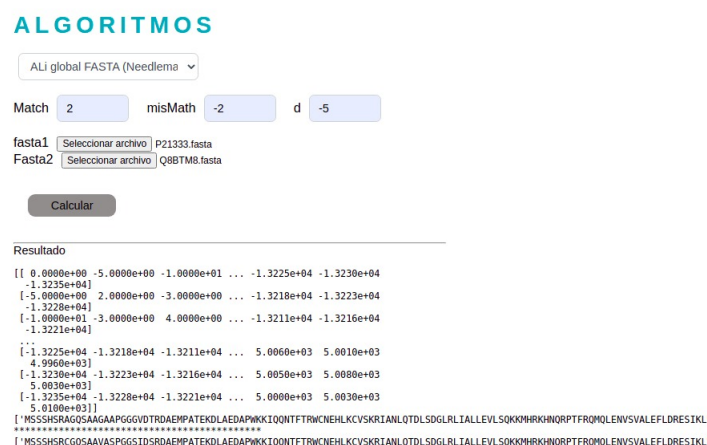
- `python3 alineamiento_local.py 'AAG' 'ACG' '2' '-2' '-2'`
- `python3 alineamiento_local.py 'archivo.fasta' 'archivo2.fasta' '2' '-2' '-2'`

El orden de los numeros es match mismatch y gap se tienen que ingresar como texto.



The screenshot shows a web interface for the Needleman–Wunsch algorithm. It has a header with 'ALGORITMOS' and 'INTEGRANTES'. Under 'ALGORITMOS', there is a dropdown menu set to 'ALI global FASTA (Needleman–Wunsch)'. Below this are input fields for 'Match' (2), 'misMath' (-2), and 'd' (-5). There are also fields for 'fasta1' and 'fasta2' with file selection buttons. A 'Calcular' button is present. The 'Resultado' section shows two sequences: 'Veca : IRSSSHSRAGGSAAGAAPGGGVDTRDAEPATERDLAEDAPMKKIQNTFTTRWCNEHLKCVSKRIANLQTDL' and 'Vecb : IRSSSHSRRCGSAAYASPGGSDSRDAEPATERDLAEDAPMKKIQNTFTTRWCNEHLKCVSKRIANLQTDL'.

Figura 1: Salida del algoritmo Needleman–Wunsch



The screenshot shows the same web interface as Figure 1, but the 'Resultado' section displays a distance matrix. The matrix is a 2x2 grid of values: [[0.0000e+00, -5.0000e+00, -1.0000e+01, ...], [-5.0000e+00, 2.0000e+00, -3.0000e+00, ...], [-1.0000e+01, -3.0000e+00, 4.0000e+00, ...], [-1.3221e+04, -1.3216e+04, -1.3211e+04, ...]]. Below the matrix, there are two sequences: 'Veca : IRSSSHSRAGGSAAGAAPGGGVDTRDAEPATERDLAEDAPMKKIQNTFTTRWCNEHLKCVSKRIANLQTDLSGGLRLIALLEVLSSQKXMRKHNRPTFRQMLENVSALEFLDRESIKL' and 'Vecb : IRSSSHSRRCGSAAYASPGGSDSRDAEPATERDLAEDAPMKKIQNTFTTRWCNEHLKCVSKRIANLQTDLSGGLRLIALLEVLSSQKXMRKHNRPTFRQMLENVSALEFLDRESIKL'.

Figura 2: Salida del algoritmo Needleman–Wunsch

2. Alineamiento local (Smith-waterman)

ALGORITMOS

Alineamiento local (Smith-wa ▼)

s1 s2 Match misMath

d

Resultado

```
[[0. 0. 0. 0.]
 [0. 2. 2. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 2.]]
Score: 2
Alineamientos:
AA
AA
```

Figura 3: Salida del algoritmo Smith-waterman

```
junior@loquipc:~/UNSA/noveno/bio/examen/algoritmos$ python3 local.py 'AAG' 'ACG'
'2' '-2' '-2'
[[0. 0. 0. 0.]
 [0. 2. 2. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 2.]]
Score: 2
Alineamientos:
AA
AA
```

Figura 4: Salida en consola

3. BLAST

Busca alineamientos de secuencias de alto puntaje entre la secuencia de consulta y las secuencias en el banco de datos usando un enfoque heurístico. BLAST requiere dos secuencias como entrada una secuencia de consulta (QUERY) y una base de datos de secuencias.

```

+ Descargas g++ blast.cpp -o blast -std=c++17
+ Descargas ./blast

Nuevo

Palabra encontrada: TGA

Tamano: 3
[3]
    T _ T
    G _ G
    A _ A

[-1]
    V _ G
    T _ T
    G _ G
    A _ A
    S _ G

[-5]
    A _ C
    V _ G
    T _ T
    G _ G
    A _ A
    S _ G
    Y _ T

V _ G
T _ T
G _ G
A _ A
S _ G

EXITO

```

Figura 5: Salida inmediata del algoritmo BLAST.

ALGORITMOS

BLAST

Matriz query.txt

Calcular

Resultado

Palabra encontrada: TGA

Tamano: 3

```

[3]
    T _ T
    G _ G
    A _ A

[-1]
    V _ G
    T _ T
    G _ G
    A _ A
    S _ G

[-5]
    A _ C
    V _ G
    T _ T
    G _ G
    A _ A
    S _ G
    Y _ T

V _ G
T _ T
G _ G
A _ A
S _ G

```

Figura 6: Salida del algoritmo BLAST.

4. Alineamiento multiple MUSCLE o CLUSTAL

Para realizar esta sección se siguieron los siguientes pasos:

- Se descarga linux 64 del siguiente link <http://www.drive5.com/muscle/downloads.htm>
- Se descomprime el .tar y luego se le da permisos para poder ejecutarse `chmod +x /usr/bin/muscle6.0.98_i86linux32`
- Para ejecutarlo se usa el siguiente comando: `home/loudev/Descargas/muscle3.8.31_i86linux64 -in '/home/loudev/Descargas/P21333.fasta' -out '/home/loudev/Descargas/output.fasta'`
- Le pasamos dos archivos uno de entrada y el archivo donde se va a guardar la salida

Para ejecutar el programa se ingresa el path completo del archivo ejecutable (o añadir el path a la consola). La salida inmediata puede ser vista en la Figura 7 y los resultados almacenados en el archivo output.fasta pueden ser visualizados en la Figura 11.

```
(bio) + Neighbor joining /home/loudev/Descargas/muscle3.8.31_i86linux64 -in "/home/loudev/Descargas/P21333.fasta" -out "/home/loudev/Descargas/output.fasta"
MUSCLE v3.8.31 by Robert C. Edgar
http://www.drive5.com/muscle
This software is donated to the public domain.
Please cite: Edgar, R.C. Nucleic Acids Res 32(5), 1792-97.
P21333 1 seqs, max length 2647, avg length 2647
(bio) + Neighbor joining
```

Figura 7: Salida inmediata al ejecutar el algoritmo muscle.

- python3 muscle_alingment.py 'P21333.fasta'

```
>sp|P21333|FLNA_HUMAN Filamin-A OS=Homo sapiens OX=9606 GN=FLNA PE=1 SV=4
MSSSHSRAGQSAAGAAPGGGVDRDAEMPATEKDLAEDAPWKKIQNTFTTRWCNEHLKCV
SKRIANLQTDLSGLRLIALLEVLSSQKMKHRKHNRPTFRQMQLENVSALEFLDRESIK
LVSIDSKAIVDGNLKLILGLIWLILHYSISMPMWDEEEDAEKKQTPKQRLLGWIQNK
PQLPITNFSRDWQSGRALGALVDSACPLCPDWSDWASKPVTNAREAMQQADDWLGIPIQ
VITPEEIVDPNVDEHSVMTYLSQFPKAKLKPGLRPLNPKKARAYGPGIEPTGNMVKK
RAEFTVETRSGAGQGEVLVYVEDPAGHQEEAKVTANNDKNRTFSVWYVPEVTGTHKVTVL
AGQHIKSPFEVYVDKSGQDASKVTAQGPGLPSGNIANKTTYFEIFTAGAGTGEVEVVI
QDPMGQKGTVEPQLEARGDSTYRCSYQPTMEGVHTVHTFAGVPIPRSPYTVTVGQACNP
SACRAVGRGLQPKGVRVKETADFKVYTKGAGSGELKVTVKGPKEERVVKQDLGDGVYGF
EYYPMPVPGTYIVITWGGQNIIGRSPFEVKVGTCEGNQKVRWAGPGLGEGVVGKSAFVVE
AIGDDVGLTGFVSGPSQAKIECDDKGDGSCDVRYPQEAAGEYAVHVLCSNEDIRLSPFM
ADIRDAPQDFHDPDRVKARGPGLKTKGDAVRDVIDIDHDNTYTVKYTPVQQGPVGVNVTYGGDPIPKSP
ALVKDNGNGTYSYVPRPKVKHTAMVSWGGVSIPNSPFRVNVGAGSHPNKVKVYGPVGA
KTGLKAHEPTYFTVDCAEAGQGDVSIKICAPGVVGPAAEDIDFDIIRNDNTFTVKYTP
RGAGSYTIMVLFADQATPTSPIRVKVEPSHDASKVKAEGPGLSRTGVELGKPTHTFTVNAK
AAGKGKLDVQFSGLTGDAVRDVIDIDHDNTYTVKYTPVQQGPVGVNVTYGGDPIPKSP
FSVAVSPSLDLKIKVSGLGEKVDVGDQEFVTKSGAGGQGVKASKIVGPSGAAPVCKV
EPGLGADNSVVRFLPREEGPYEVEVTDGVPVPGSPFLEAVAPTSPKVKAFGPGQLQGG
SAGSPARFTIDTKGAGTGGLGLTVEGPGCEAQLCELDNGDGTCSVSYVPTGPDYNINILF
ADTHIPGSPFKAHVVPFCFASKVKCSGGLERATAGEVGQFQVDCSSAGSAELTIEICSE
AGLPAEVYIQDHGDGHTHTITYIPLCPGAYTVTIKYGQPVNPFPSKLQVEPAVDTSGVQC
YGPGLGEGGVFREATTEFSVDARALTQTGGPHVKARVANPSGNLTETVYQDRGDGMKYVE
YTPYEEGLHSVDVTDGSPVSPSPFQVPVTEGCDPSRVRVHGPQISGTTNKNPKFTVET
RGAGTGGLGLAVEGPSEAKMSCMDNDGSCSVYIPYAGTYSLNVTYGGHQVPGSPFKV
PVHDVTDASKVKCSGGLSPGMVRANLPQSFQVDTSKAGVAPLQVQVQGPGLVEPVDV
DNADGTQVNYVPSREGPYSISLVYGDVEVPRSPFKVKVLPHTDASKVKAAGSGPLNTTG
PASLPVEFTIDAKDAGEGLLAVQITDPEGPKKTHIQDNHDGTYTVAYVPDVTGRYTI
KYGGDEIPFSPYRVAVPTGDASKCTVTVSIGGHLGAGIGPTIQIETEVTITVDTKAAG
KKGVTCTVCTPDGSEVDVDVVENEDGTDFIFYTAPQPGKYVICVRFGEHVPNSPFQVTA
LAGDQPSVQPLRSQQLAPQYTYAQQGQQTWAPERPLVGVNGLDVTSLRPFDLVIPFTIK
KGEITGEVRMPSGKVAQPTITDNKDGTVTVRYAPSEAGLHEMDIRYDNMHIHPSPLQFV
DYVNCGHVTAYGPGLTHGVNKPATFTVNTKDAGEGGLSLAIEGPSKAEISCTDNQDGTG
```

Figura 8: Salida almacenada en el archivo output.fasta despues de ejecutar el algoritmo Muscle sobre la secuencia humana P21333.fasta.

ALGORITMOS

MUSCLE o CLUSTAL

fasta1 P21333.fasta

Resultado

```
>sp|P21333|FLNA_HUMAN Filamin-A OS=Homo sapiens OX=9606 GN=FLNA PE=1 SV=4
MSSSHSRAGQSAAGAAPGGGVDTRDAEMPATEKDLAEDAPWKKIQNTFTRWCNEHLKCV
SKRIANLQTDLSGDLRLIALLEVLQKKMHRKHNRPTFRMQLENVSVALEFLDRESIK
LVSIDSKAIVDGNLKLILGLIWTLLHYSISMPMWDEEEDAEAKKQTPKQRLGWIQNK
PQLPITNFSRDWQSGRALGALVDSAPGLCPDWSDWASKPVTNAREAMQADDWLGIPO
VITPEEIVDPNVDEHSVMTYLSQFPKAKLKPGLRPKLNPKKARAYGPGIEPTGNMVKK
RAEFTVETRSAGQGEVLVYVEDPAGHQEEAKVTANNDKNRTFSVWYVPEVTGTHKVTVLF
AGQHIASPFVEYVDKSSQGDASKVTAQGGPLEPSGNIANKTTYFEIFTAGAGTGEVEVVI
QDPMGQKGTVEPQLEARGDSTYRCSYQPTMEGVHTVHTFAGVPIPRSPYTVTVGQACNP
SACRAVGRGLQPKGVVRKETADFKVYTKGAGSGELKVTGKPGKEERVVKQDLGDGVYGF
EYYPMPGTYIVTITWGGQNIGRSPFEVKVGTCEGNQKVRWGPGLGGVVGKSADFVVE
AIGDDVTGLGFSVEGPSQAKIECDDKGDGSCDVRYPQEAEGYAVHVLCSNEDIRLSPFM
ADIRDAPODFHPDRVKARGPGLKGTGVAVNKPAEFTVDAKHGGKAPLRVQVQDNEGCPVE
ALVKDNGNGTYSYVPRKPVKHTAMVSWGGVSIPNSPFRVNVGAGSHPNKVYVGPVGA
KTGI KAEHPTVFTVNCFAAGGNGVSTGTGAPGVVGPFAFNTDFTTRNDNTFTVKYTP
```

Figura 9: Salida del algoritmo Muscle o Clustal

5. **Modelos de sustitución Jukes-cantor y Kimura** En biología, los modelos de sustitución, también llamados modelos de evolución de la secuencia de ADN, son modelos de Markov que describen los cambios a lo largo del tiempo evolutivo. Estos modelos describen los cambios evolutivos en las macromoléculas (por ejemplo, las secuencias de ADN) representadas como secuencias de símbolos (A, C, G y T en el caso del ADN). Los modelos de sustitución se utilizan para calcular la probabilidad de los árboles filogenéticos utilizando datos de alineación de secuencias múltiples permitiendo una estimación verosímil.

ALGORITMOS

Kimura

fasta1 P21333.fasta
fasta2 Q8BTM8.fasta

Resultado

Kimura
-0.0

Figura 10: Salida del algoritmo Kimura

ALGORITMOS

Jukes-cantor

seq 1
AGC
seq 2
ACG

Calcular

Resultado

```

numero de coincidencias
1
numero de inconsistencias
2
Proporción entre desajustes y longitud de secuencia
0.6666666666666666
Jukes-Cantor formula  $K = (-3/4)\ln(1-(4/3)B)$ 
La distancia molecular entre ambas secuencias es:
1.6479184330021643

```

Figura 11: Salida del algoritmo Jukes-cantor

6. UPGMA

Método de clustering que se basa en la identificación de las parejas más similares y cálculo de la media de las distancias entre ellas y el resto de las secuencias para reconstruir el árbol.

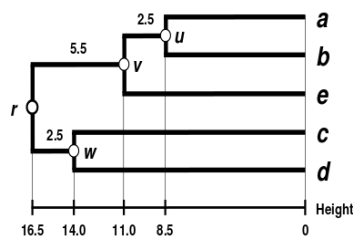


Figura 12: Ejemplo

El algoritmo UPGMA recibe dos parámetros de entrada una matriz y sus etiquetas.

```
pejelagarta@pejelagarta-FX503VD: ~/Escritorio/BIOINFORMA...
pejelagarta@pejelagarta-FX503VD:~/Escritorio/BIOINFORMATICA$ python UPGMA.py
(((A,D),((B,F),G)),C),E)
pejelagarta@pejelagarta-FX503VD:~/Escritorio/BIOINFORMATICA$ python3
Python 3.8.10 (default, Jun 2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import newick
>>> tree= newick.loads('(((A,D),((B,F),G)),C),E')[0]
>>> print(tree.ascii_art())
      A
     /
    /
   /
  /
 /
/
E
 \
  \
   \
    \
     \
      C
     /
    /
   /
  /
 /
/
G
 \
  \
   \
    \
     \
      F
     /
    /
   /
  /
 /
/
B
 \
  \
   \
    \
     \
      D
     /
    /
   /
  /
 /
/
A
>>>
```

Figura 13: Salida del algoritmo UPGMA

ALGORITMOS

UPGMA

label A-G

Matriz matriz_upgma.txt

Calcular

Resultado

((G,D),((F,B),((E,A),C)))

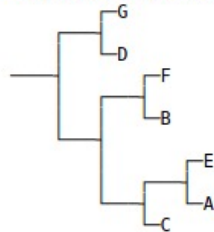
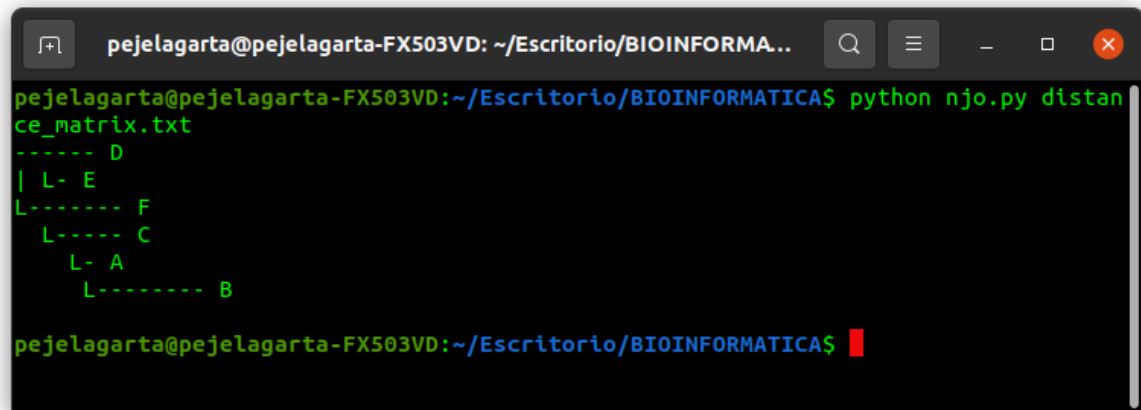


Figura 14: Salida del algoritmo UPGMA

7. Neighbor Joining



```
pejelagarta@pejelagarta-FX503VD: ~/Escritorio/BIOINFORMA...
pejelagarta@pejelagarta-FX503VD:~/Escritorio/BIOINFORMATICAS$ python njo.py distance_matrix.txt
----- D
| L- E
L----- F
  L----- C
    L- A
      L----- B
```

Figura 15: Salida del algoritmo Neighbor Joining

ALGORITMOS

Neighbor Joining

Matriz distance_matrix.txt

Calcular

Resultado

```
----- D
| L- E
L----- F
  L----- C
    L- A
      L----- B
```

Figura 16: Salida del algoritmo Neighbor Joining

REPOSITORIO

El repositorio de la página creada con los algoritmos implementados se encuentra en <https://github.com/junidev/bioinformatica/tree/master>